Hochschule Mittweida (FH)
University of Applied Sciences

MITTWEIDA
ROSSWEIN

Fachbereich Medien

Mücke, Steffen

Auf dem Weg zum automatisierten Online-Shop – Grundlagen der Integration und Umsetzung eines Prototyps zur Anbindung von Warenwirtschaftssystemen an Magento Commerce

– Diplomarbeit –

Hochschule Mittweida – University of Applied Sciences (FH)

Mittweida – 2009

Hochschule Mittweida (FH)
University of Applied Sciences

MITTWEIDA
ROSSWEIN

Fachbereich Medien

Mücke, Steffen

Auf dem Weg zum automatisierten Online-Shop – Grundlagen der Integration und Umsetzung eines Prototyps zur Anbindung von Warenwirtschaftssystemen an Magento Commerce

– eingereicht als Diplomarbeit –

Hochschule Mittweida – University of Applied Sciences (FH)

vorgelegte Arbeit wurde verteidigt am: _____

Erstprüfer	Zweitprüfer
Prof. Dr. phil. Ludwig Hilmer	Dipl.-Ing. Michael Ablass

Datum der Abgabe: _____

Mittweida – 2009

Bibliografische Beschreibung

Mücke, Steffen:

Auf dem Weg zum automatisierten Online-Shop – Grundlagen der Integration und Umsetzung eines Prototyps zur Anbindung von Warenwirtschaftssystemen an Magento Commerce. – 2009 – 127 S.

Mittweida, Hochschule Mittweida (FH), Fachbereich Medien, Diplomarbeit

Kurzreferat

Die Diplomarbeit beschäftigt sich mit der Integration verteilter Unternehmensanwendungen. Zunächst werden Kernaspekte zu Problemdimensionen, Kategorisierungen und Methoden der Integration dargestellt. Nach einer Darstellung oft verwendeter Integrationstechniken unter besonderer Berücksichtigung von Web Services wird ein konkretes Integrationsszenario aus dem Bereich des E-Commerce dargestellt. Es umfasst die Integration des E-Commerce-System Magento Commerce mit Warenwirtschaftssystemen allgemein und Alea Commerce Suite als konkreten Vertreter. Im weiteren Verlauf der Arbeit werden die Schnittstellen der Anwendungen analysiert und es wird eine Integrationslösung erstellt, und deren wesentliche Funktionen und Methodiken dargelegt. Am Ende der Arbeit ist ein funktionsfähiger Prototyp der Integrationslösung entstanden.

Danksagung

Für die empfangene Unterstützung, die gute Zusammenarbeit, für Tipps, Ratschläge und Beistand in der Not möchte ich mich an dieser Stelle herzlich bedanken bei:
meinen Betreuern Prof. Ludwig Hilmer und Michael Ablass,
Helga Trölenberg-Buchholz von ALEA,
dem kompletten Team von Netresearch, besonders Stephan, Christian und Thomas,
Rico Neitzel und Roy Rubin, stellvertretend für die Magento-Community.

Außerdem danke ich all den kleinen und großen Helferlein, guten Geistern, Engeln und Bengeln, Zwei-, Drei- und Vierbeinern, Bekannten und Unbekannten, die mich in jeder Hinsicht unterstützt und ertragen, getrieben und geschont, entmutigt und ermutigt und überhaupt erfreut haben in dieser Zeit. Besonders aber bei: Anke-Jenny, Sven, Tarik, Dominic, Daisy, Erik (der Wikinger), Anouk, Norman, George, Tobias, Gecko und Waran.

Und ganz besonders danke ich meiner gesamten Familie. Ihr seid die Besten.

Was als ein Strom nützlicher Informationen begann,
hat sich inzwischen in eine Sturzflut verwandelt.
Neil Postman, amerik. Medienkritiker

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
bspw.	beispielsweise
bzw.	beziehungsweise
CSV	Comma Separated Values
CLI	Command Line Interface
CGI	Common Gateway Interface
DBMS	Datenbank-Management-System
DCE	Distributed Computing Environment
d.h.	das heißt
DNA	Deoxyribonucleic Acid
DTD	Document Type Definition
EAI	Enterprise Application Integration
EBXML	Electronic Business using XML
etc.	et cetera
ETL	Extract, Transform, Load
FDBMS	Föderiertes Datenbank-Management-System
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
ISO	Internationale Standardisierungs-Organisation
IT	Informations-Technik
OASIS	Organization for the Advancement of Structured Information Standards
OSI	Open Systems Interconnection
o.ä.	oder ähnliche

PHP	Hypertext Preprocessor
RPC	Remote Procedure Call
s.	siehe
S.	Seite
SCM	Supply Chain Management
SGML	Standard Generalized Markup Language
SOAP	Eigenname seit Version 1.2, ursprünglich Simple Object Access Protocol
SQL	Structured Query Language
TCP	Transmission Control Protocol
u.	und
u.a.	unter anderem; auch: und andere
u.ä.	und ähnliche
übers.	übersetzt
UDDI	Universal Description, Discovery and Integration
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
usw.	und so weiter
v.a.	vor allem
vgl.	vergleiche
VM	Virtual Machine
W3C	World Wide Web Consortium
WaWi	Warenwirtschaft; auch Warenwirtschaftssystem
wörtl.	wörtlich
WS	Web Services
WSDL	Web Services Description Language
XML	Extensible Markup Language
z.B.	zum Beispiel
ZF	Zend Framework

1 Einleitung

1.1 Hinführung

Die Integration verteilter Anwendungen zählt schon seit Jahren zu den wichtigsten Themen in der Unternehmens-IT und gewinnt noch immer an Relevanz. Nahezu alle Unternehmensbereiche setzen umfassende IT-Systeme ein, um ihre Geschäftsprozesse zu unterstützen. Um im Wettwerb und unter fortlaufend sich änderndern Marktbedingungen bestehen zu können, müssen die Unternehmen sich anpassen und weiterentwickeln. Neue Unternehmensbereiche benötigen fast immer neue, eigene IT-Systeme. Die alten Systeme werden in der Regel weiterhin benötigt und bleiben zentraler Bestandteil der IT-Landschaft. So wächst das Anwendungsportfolio eines Unternehmens stetig an. Viele Teilsysteme benötigen Informationen aus anderen Teilsystemen oder liefern Informationen für diese. Interaktionen und Vernetzungen müssen geschaffen werden, wo Teilsysteme miteinander kommunizieren sollen. Systeme, die so nicht für eine Kooperation untereinander entwickelt wurden, müssen zusammenarbeiten; sie müssen *integriert* werden. Es gibt Schätzungen, die besagen, dass heute mehr als die Hälfte aller IT-Kosten auf die Integration existierender Systemen aufgewendet werden [Leser / Naumann, 2007, 1].

Grundziele von Integrationsvorhaben sind zum Einen die Schaffung einer übergeordneten Sicht auf Informationen aus verteilten Anwendungen, und zum Anderen die Einbindung von bereits bekanntem Wissen oder bereits existenter Funktionalität in einen neuen Anwendungskontext. Gesucht wird die optimale Kombination der verhandenen Einzelsysteme zum Zwecke einer höheren Produktivität und Effektivität. Material-, Geld- und Informationsflüsse sollen in zeitlicher, qualitativer und finanzieller Hinsicht optimiert werden. Unnötige Medienbrüche sind zu vermeiden. Ein möglichst großer Teil der Informationen kann automatisch, d.h. ohne menschliche Intervention an die jeweils relevanten Empfänger übermittelt und verarbeitet werden. Integrierte Informationen von allen Prozessen entlang der gesamten Wertschöpfungskette eines Unternehmens sollen Entscheider bei ihrer Aufgabe unterstützen. Letztlich ist der *Return on Investment* (ROI), die Rentabilität des eingesetzten Kapitals, zu steigern. Dabei müssen aber jederzeit die *Kunden* im Mittelpunkt stehen. Die Kundenorientierung und Kundenzufriedenheit sind neben dem ROI die zweite Grundsäule des modernen integrierten Unternehmens.

Im Bereich des E-Commerce, des elektronischen Handels über das Internet, haben sich zwei Gruppen von Standardsystemen entwickelt, die u.a. im Hinblick auf die Integrationsmöglichkeiten deutliche Unterschiede aufweisen. Zum Einen gibt es mächtige, kommerziell vertriebene E-Commerce-Systeme, die vor allem für größere Unternehmen relevant sind. Bekannte Vertreter sind *Inter-*

shop oder *Oxid*. Diese Systeme basieren auf Systemplattformen wie Java EE oder .NET und bieten vielfältige und ausgereifte Integrationsmöglichkeiten im Lieferumfang, sind jedoch in der Anschaffung und im Unterhalt so teuer, dass sie sich für kleine und mittlere Unternehmen in der Regel nicht lohnen. Zum Anderen existiert eine Anzahl von E-Commerce-Systemen, die eher auf diese kleineren Unternehmen fokussiert sind. Diese Systeme sind meist in webbasierten Skriptsprachen (u.a. → PHP und Perl) entwickelt und sind wesentlich günstiger oder sogar als kostenlose Open-Source-Systeme erhältlich, wie z.B. die in Deutschland recht verbreiteten Systeme *osCommerce* und *xt:Commerce*. Der Funktionsumfang solcher Systeme ist allerding wesentlich geringer als bei der erstgenannten Gruppe; vor allem auch im Bereich der Integrationsmöglichkeiten bestehen klare Beschränkungen. Solche Systeme müssen für die Umsetzung von konkreten Integrationsvorhaben zumeist im Einzelfall mit viel Aufwand angepasst und erweitert werden.

Zwischen den genannten Gruppen existiert eine Angebotslücke. Es fehlen Anwendungen, die auch für kleine und mittelgroße Unternehmen attraktiv sind, und ihnen gleichzeitig genügend professionelle Funktionen und Integrationsmöglichkeiten bieten, um ihre Anwendungsstruktur effizient integrieren zu können. Seit einiger Zeit liegt mit dem E-Commerce-System *Magento Commerce* der amerikanischen Softwarefirma Varien ein vielversprechender Vertreter dieser neuen Gruppe von E-Commerce-Systemen vor. Magento ist ein komplett neu entwickeltes Open-Source-System, das sich jedoch in seinem Funktionsumfang eher an kommerziellen Systemen für große Unternehmen orientiert. Es bietet neben einer Vielzahl neuartiger Features auch einige für die Integration sehr interessante Schnittstellen, u.a. eine → SOAP Api. Mit Magento wird es vor allem für mittelständische Unternehmen[1] erheblich leichter, die SCM-orientierte Integration auch im Bereich des E-Commerce zu betreiben und von deren Vorteilen zu profitieren. Laut [Wegweiser GmbH, 2007, 12] haben diese in Hinsicht auf die SCM-orientierte Integration durchaus noch Nachholbedarf.

Durch die Tätigkeit bei der Firma Netresearch GmbH & Co. KG, zu deren Geschäftsschwerpunkten die Bereiche E-Commerce mit Magento Commerce und Content Management Systeme gehören, ist der Verfasser frühzeitig mit Magento in Kontakt gekommen. Konkreter Bedarf nach einer Integrationslösung beim Einsatz von Magento wurde wiederholt von Kunden an die Firma herangetragen. Die Möglichkeit der Zusammenarbeit von Netresearch mit der

[1] Nach [Reichling, 2001, 95] soll in dieser Arbeit unter einem mittelständischen Unternehmen eine wirtschaftliche Entscheidungseinheit verstanden werden, „die eine im Vergleich zu anderen Entscheidungseinheiten relativ geringe Größe aufweist, und vom Eigentümer-Unternehmer unter der Maßgabe geführt wird, die Entscheidungseinheit in Eigenverantwortung zu halten“.

Softwarefirma ALEA mit Sitz in Jena, die derzeit ein neues Warenwirtschaftssystem mit Fokus auf mittelständische Versandhandelsunternehemen einführt, war ebenfalls eine Motivation für diese Arbeit.

1.2 Zieldefinition

Das Ziel der vorliegenden Arbeit ist die prototypische Umsetzung einer Integrationslösung für die Anbindung des E-Commerce-System *Magento Commerce* an Warenwirtschaftssysteme. Als konkreter Vertreter eines Warenwirtschaftssystems wurde *ALEA Commerce Suite* ausgwählt. Die Integrationslösung soll jedoch prinzipiell auch die Anbindung anderer Warenwirtschaftssysteme (z.B. Microsoft Dynamics) ermöglichen.

Die Integrationslösung soll in erster Linie den Nachweis erbringen, dass die neuen Schnittstellen in Magento Commerce für Integrationsaufgaben dieser Art zu verwenden sind. Folgende Funktionalitäten soll die Integrationslösung zu diesem Zweck umfassen:

- Abgleich und Verwaltung der Produktdaten
- Abgleich und Verwaltung von Kundendaten
- Automatische Weitergabe von Bestellvorgängen
- Abgleich und Verwaltung der Katalog- und Kategoriedaten

Prototypische Umsetzung bedeutet für diese Arbeit, dass die Integrationslösung die prinzipielle Machbarkeit und Umsetzung verwirklicht, ohne jedoch bereits als fertige Anwendung für den Einsatz in realen Produktivsystemen gelten zu können.

Weitere Ziele der Arbeit sind:

- eine überblickshafte Darstellung der wichtigsten Konzepte, Ansätze und Methoden von Integration, die als umsetzungsorientierte Einführung in das Thema dienen kann, sowie
- die Dokumentation der Anbindungsmöglichkeiten weiterer Warenwirtschaftssysteme.

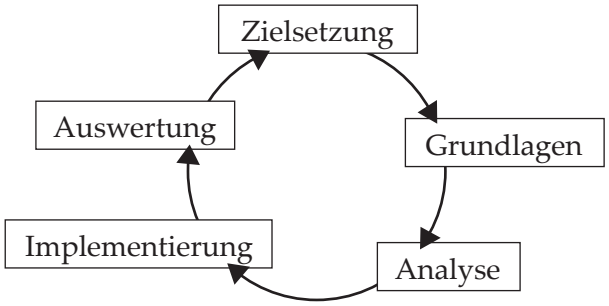

Abbildung 1.1: Aufbau der Arbeit. Die Darstellung als Iterationsmuster soll verdeutlichen, dass das Ergebnis selbst nur Ausgangspunkt weiterer Überlegungen und Arbeiten vor einem Produktiveinsatz ist.

1.3 Aufbau der Arbeit

Die Arbeit gliedert sich in fünf Kapitel. Nach der Einleitung folgt zunächst ein umfangreicher Überblick über das Themenfeld Integration. Es werden grundlegende Ansätze, Methoden und Kategorisierungen vorgestellt, die das Thema gliedern und eine Einordnung des konkreten Integrationsfalls ermöglichen. Es wird auf wesentliche in der Praxis verbreitete Umsetzungstechnologien eingegangen.

Darauf aufbauend wird im dritten Kapitel der konkrete Integrationsfall betrachtet. Die beteiligten Komponenten werden vorgestellt, ihre Eignung und Schnittstellen analysiert, und anschließend der Umsetzungsansatz der Integrationslösung abgeleitet. Die Ergebnisse der Analyse bilden die Grundlage für die Umsetzung der Integrationslösung. Die konkrete Implementierung, das Entwerfen und Gestalten des Quellcodes, ist dagegen nur am Rande Gegenstand dieser Arbeit.

Im vierten Kapitel werden einige Implementierungsaspekte der Integrationslösung näher vorgestellt. Die betrifft vor allem die Automatisierung und die Organisation des Datenaustauschs über Dateien. Außerdem werden die implementierten Funktionalitäten präsentiert, sowie die Anbindungsmöglichkeiten weiterer Warenwirtschaftssysteme dokumentiert. Der grundlegende Aufbau der Programmierlösung wird ebenfalls dargestellt.

Das abschließende fünfte Kapitel fasst die erreichten Ziele zusammen, analysiert die erreichte Leistungsfähigkeit der Integrationslösung und endet mit einem Ausblick auf die nötigen weiteren Arbeiten und Verbesserungen, die auf dem Weg zu einem Einsatz in einer Produktivumgebung nötig sind.

2 Grundlagen der Integration

Dieses Kapitel behandelt die Grundlagen des Themenbereich Integration in der Unternehmens-IT. Zunächst werden einige Grundbegriffe eingeführt, auf denen die weitere Arbeit aufbaut. Im Anschluß sind die Zielsetzungen, Anwendungsfelder und Problemdimensionen der Integration zu diskutieren. Im Weiteren werden die konkreten Anforderungen an eine Integrationslösung aus technischer und betriebswirtschaftlicher Sicht herausgearbeitet, sowie verschiedene Integrationsansätze und Integrationsmethoden vorgestellt. Den Abschluss des Kapitels bildet eine kurze Gegenüberstellung einiger in der Praxis verbreiteter Integrationstechniken. Der Umfang des Themas erlaubt keine in allen Punkten erschöpfende Darstellung im Rahmen dieser Arbeit. An verschiedenen Stellen wird daher auf weiterführende Fachpublikationen zum jeweiligen Thema verwiesen werden.

2.1 Grundbegriffe

2.1.1 Informationen und Daten

Die Begriffe *Informationen* und *Daten* werden häufig synonym gebraucht, sind es aber nicht. Daten sind Träger von Informationen. In Daten werden Informationen auf eine spezifizierte Weise hinterlegt (kodiert). Die jeweilige Art der Kodierung kann jedoch sehr unterschiedlich ausfallen, und ist nicht immer durch Menschen lesbar (z.B. Binärdaten). Grundsätzlich kann festgehalten werden:

Informationen sind konkrete, atomare Aussagen zu einem beliebigen, definierten Sachverhalt, die für den menschlichen Geist verständlich sind und dem Empfänger eine Bedeutung vermitteln. Informationen bewirken eine Wissensänderung beim Empfänger und sind daher zweckorientiert. Weiterhin sind sie immateriell und daher immer an ein Trägermedium - die Daten - gebunden. [vgl. Heine, 1999, 25]

Daten sind Träger von nach spezifischen Regeln kodierten Informationen, die zwingend für Maschinen – jedoch nicht zwingend für Menschen – verständlich und auswertbar sind. Daten bedürfen der *Interpretation*, um die enthaltenen Informationen nutzbar zu machen. „Sie sind dargestellte Information, nicht selbst Information“ [Heinrich u. a., 2004, 166].

In der Literatur gibt es eine Vielzahl von Definitionen des Datenbegriff, die sich grundlegend in zwei Ansätze unterteilen lassen [Heinrich u. a., 2004, 166f]:

1. Daten als maschinell verarbeitbare Zeichen (syntaktischer Ansatz)

2. Daten als eine Abbildung von Objekten, d.h. Gegenständen, Personen, Zuständen und Vorgängen der realen Welt oder der Vorstellungswelt des Menschen (semantischer Ansatz)

Das folgende Beispiel veranschaulicht den Unterschied zwischen Informationen und Daten: Die Sätze „Der Künstler malt ein Bild“ und „The artist paints a picture“ stellen unterschiedliche Daten (Zeichenfolgen) dar, beinhalten aber dieselbe Information, während das Datum „ein großer Künstler“ aus physikalischer bzw. intellektueller Sicht unterschiedliche Informationen beinhaltet.

In dieser Arbeit wird im Folgenden der zweite, semantische Definitionsansatz verwendet. Er repräsentiert den Datenbegriff in für die menschliche Auffassung besser geeigneter Form, da er auf die Bedeutung der Daten abzielt.

Anhand des Strukturierungsgrades der Daten lassen sich drei Klassen von Daten, bzw. Datenformaten unterscheiden [vgl. Leser / Naumann, 2007, 17; Lohse, 2003, 7]:

Strukturierte Daten besitzen ein Schema, welches ihre Struktur verbindlich festlegt und wesentlich zur Bedeutung der Daten beiträgt. Durch das Schema ist jedes Datenfeld eindeutig definiert und adressierbar. Der verbreitetste Vertreter strukturierter Daten ist die Datenbank.

Semistrukturierte Daten sind ebenfalls nach einem grundlegenden Schema aufgebaut, können aber auch von diesem abweichen. Insbesondere können Datenfelder zwar durch das Schema definiert, aber in den einzelnen Datensätzen nicht, oder auch mehrfach vorhanden sein [Bry u. a., 2005]. Semistrukturierte Daten werden auch selbsterklärende Daten genannt, da sie ihr Schema in den Daten selbst definieren und transportieren. Der wichtigste Vertreter sind XML-Dateien.

Als *unstrukturierte Daten* werden Daten ohne ein Datenschema bezeichnet. Die Struktur der dargestellten Informationen wird hier nicht mit den Daten gespeichert. Als beispielhafter Vertreter sind natürlichsprachliche Texte zu nennen. Die maschinelle Auswertung von unstrukturierten Daten ist meist sehr aufwändig umzusetzen, da ein umfangreiches und individuelles interpretatorisches Wissen zur Extraktion der Informationen vonnöten ist.

2.1.2 Nachrichten und Kommunikation

Damit Daten einen Nutzen entfalten können, müssen sie von einem Sender zu einem Empfänger übertragen werden. Durch die Übertragung werden Daten zu *Nachrichten*, welche die Grundlage der Kommunikation darstellen [Heine,

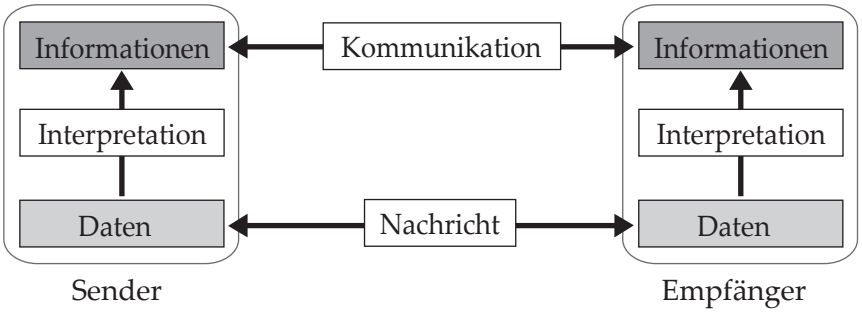

Abbildung 2.1: verdeutlicht die Zusammenhänge der Begriffe Informationen und Daten, bzw. Kommunikation und Nachricht. Erstellt nach [Heine, 1999, 26].

1999, 26]. Eine Nachricht im technischen Sinne besteht aus einem Nachrichtenkopf, dem „Header“, der z.B. die Adresse des Empfänger beinhaltet, und der eigentlichen Nachricht, dem „Body“ [Heinrich u. a., 2004, 449]. Damit eine Nachricht einen Informationsfluss, bzw. eine Wissensänderung beim Empfänger bewirkt, benötigen beide Kommunikationspartner dasselbe Wissen (dieselben Algorithmen) zur Interpretation der Nachricht.

Unter *Kommunikation* wird der Austausch von Informationen verstanden. Dieser Austausch umfasst das Übermitteln von Daten und die Gewinnung von Informationen durch Interpretation[Heine, 1999, 27]. Neben dem Inhaltsaspekt besitzt Kommunikation auch einen Beziehungsaspekt, der das Verhältnis der beteiligten Kommunikationspartner beschreibt [Heinrich u. a., 2004, 364f], bspw. die Beziehungen personeller Aufgabenträger in einem Unternehmen [Heine, 1999, 27].

2.1.3 Integration

Im allgemeinen Verständnis bezeichnet Integration „die Herstellung oder Wiederherstellung eines Ganzen durch Vereinigen oder Verbinden logisch zusammengehöriger Teile“ [Heinrich u. a., 2004, 333]. Im Kontext der betrieblichen Informationssysteme bestehen die Teile einer Integration in den verschiedenen isolierten IT-Systemen, die systemintern zu einer Integrationslösung verbunden werden sollen [Schwarze, 2000, 131]. Das angestrebte Ergebnis der Integration ist eine - wie auch immer geartete - *Steigerung der Leistungsfähigkeit des Gesamtsystems* [Voigtmann / Zeller, 2002, 8]. Diese kann sich durch verschiedene Kriterien definieren, z.B. durch die Bereitstellung neuer Funktionen oder durch die Nutzbarmachung bisher isolierter Daten. Aus unternehmerischer Sicht ist

das Ziel von Integration immer auch eine *Effizienzsteigerung* der innerbetrieblichen Abläufe, sowie die Schaffung von *Wettbewerbsvorteilen* [Speyerer, 2005, 9]. Diesen Effizienzsteigerungen steht allerdings ein hoher Investitionsaufwand für die Schaffung des Integrationssystems gegenüber. Die Nutzeffekte ergeben sich eher langfristig [Schwarze, 2000, 132].

Integrationssysteme bieten einen *transparenten Zugriff* auf die integrierten Teilsysteme [Leser / Naumann, 2007, 8]. Das bedeutet, dass der Anwender der Integrationslösung nicht weiß, und auch nicht wissen muss, aus welchem Teilsystem die angeforderten Informationen oder Funktionen stammen. Der Anwender interagiert über ein Teilsystem mit allen anderen Teilsystemen, die zur Erfüllung der jeweiligen Anforderung benötigt werden, ohne diese kennen oder bedienen zu müssen. Das Ausmaß der Transparenz kann je nach Integrationsfall differieren. Während es auf den ersten Blick erstrebenswert scheint, stets die größtmögliche Transparenz zu realisieren, ist dies nicht immer auch sinnvoll oder möglich[Leser / Naumann, 2007, 8]. So ist es z.B. angebracht, dass ein System, welches in einer Bibliothek mehrere interne und externe Literaturdatenbanken integriert, den Anwender bestimmen lässt, welche Datenquellen er für seine Suche verwenden möchte.

2.2 Ziele von Integration

In der Literatur werden die Ziele von Integration grundlegend in zwei Kategorien aufgeteilt: in *unternehmerische Ziele* und *technische Ziele* [vgl. z.B. Heine, 1999; Lohse, 2003; Speyerer, 2005]. Im folgenden wird die Kategorisierung erweitert um allgemeine Ziele, deren Umsetzung sowohl von der technischen als auch von der unternehmerischen Ebene her Rechnung getragen werden muss.

2.2.1 Allgemeine unternehmerische Ziele

Unter den allgemeinen unternehmerischen Zielen sind diejenigen Ziele verortet, deren Umsetzung auf allen Ebenenen der an einer Integrationslösung Beteiligten Rechnung getragen werden muss. Zu den allgemeinen unternehmerischen Zielen gehören [vgl. Heine, 1999, 81f; Lohse, 2003, 11; Speyerer, 2005, 15f]:

Positive Rentabilität: Der durch das Integrationssystem erzielte Gewinn muss die Höhe des eingesetzten Kapitals überschreiten, damit sich das System überhaupt lohnt. Zum eingesetzten Kapital sind dabei neben der Entwicklung des

Systems und der Anschaffung der benötigten Hard- und Software auch Kosten für Personalschulung, Wartung, Beratungsleistungen, zukünftig anfallende Investitionen, etc. zu zählen [Speyerer, 2005, 16].

Einsparung und effektive Nutzung von Ressourcen: Durch die Rationalisierung und Automatisierung von Arbeitsabläufen sollen Geschäftsvorgänge schneller, flexibler und weniger fehlerbehaftet bearbeitet werden. Der Aufwand zur Erfassung von Daten in den verschiedenen Unternehmensbereichen wird reduziert, wenn Daten unternehmensweit verwendbar sind. Auch bei der Administration des Systems werden Einsparungsmöglichkeiten gesehen, etwa aufgrund der zahlenmäßigen Reduktion der zu betreuenden Schnittstellen und Datenquellen [Heine, 1999, 81].

Sicherung der Datenqualität: Daten sind heute das wichtigste Kapital vieler Unternehmen. Folglich müssen besonders hohe Ansprüche an die Sicherung der Datenqualität gestellt werden. Bei der Integration soll die Datenerfassung so gestaltet werden, dass die Daten schon bei der Erfassung in optimaler Qualität vorliegen. Das umfasst u.a. die Vermeidung von *Eingabefehlern* und von *Redundanzen* durch mehrfache Erfassung von Datensätzen. Die wichtigsten Dimensionen der Datenqualität sind u.a. die Relevanz, die Korrektheit, sowie die Abbildungsgenauigkeit der erfassten realen Objekte.

Bessere Informationsversorgung der Entscheidungsträger: Das Integrationssystem bietet der Unternehmensführung umfassende und einheitliche Informationen über alle beteiligten Geschäftsprozesse. Mit diesem umfassenden Wissen können Entscheidungsträger fundiertere Entscheidungen treffen und das Unternehmen besser im Wettbewerb positionieren. Sie können so Optimierungspotentiale identifizieren, die Entwicklung neuer Produkte beschleunigen, neue Geschäftsfelder erschließen, etc. [vgl. Speyerer, 2005, 16].

Steigerung der Kundenzufriedenheit: Die integrierte Informationssicht ermöglicht es, den Kunden jederzeit schnell und umfassend über den aktuellen Fertigungsstand des Produktes zu informieren. Zusätzlich können durch die enge Kommunikationsvernetzung der an der Fertigung beteiligten Bereiche, je nach Art des Produktes, auch sehr späte Änderungswünsche seitens des Kunden besser berücksichtigt werden [Speyerer, 2005, 16].

Erhöhung der Aktualität der Daten: Wenn sich überlagernde Daten von verschiedenen Teilsystemen erhoben werden, werden Änderungen und Ergänzungen nur im jeweils aktiven Teilsystem gespeichert. Mit der Integration soll sichergestellt werden, dass durch die einheitliche Datenbasis veraltete und inkonsistente Daten ausgeschlossen werden.

2.2.2 Technische Ziele

Die technischen Ziele sind vor allem für die IT-Mitarbeiter von Relevanz, die sicherstellen müssen, dass das Integrationssystem zuverlässig und langlebig die gestellten Anforderungen erfüllen kann. Zu den technischen Zielen gehören [vgl. Heine, 1999, 81f; Lohse, 2003, 11; Speyerer, 2005, 15f]:

Erhöhung der Datenintegrität: Der Begriff Datenintegrität fasst Maßnahmen zur Sicherung der *Datenkonsistenz*, der *Datensicherheit* und des *Datenschutzes* zusammen [Heinrich u. a., 2004, 175]. Das Ziel aller dieser Maßnahmen ist die Gewährleistung von leistungsfähigen, gültigen und sicheren Datenquellen.

Erweiterbarkeit: Das Integrationssystem muss neue Teilanwendungen flexibel einfügen können, um zukünftigen Änderungen der Geschäftsprozesse und des Anwendungsportfolios gewachsen zu sein.

Skalierbarkeit: Das Integrationssystem sollte bei Bedarf auch auch anderen Rechenumgebungen lauffähig sein und bei Zuteilung von mehr Ressourcen auch eine bessere Performance liefern.

Herstellerunabhängigkeit und *Standard-Kompatibilität* sind weitere Forderungen, die ein System flexibel, robust und lange einsetzbar halten sollen.

2.2.3 Spezielle betriebswirtschaftliche Anforderungen

Die in Abschnitt 2.2, S. 10 angeführten allgemeinen unternehmerischen Ziele stellen bereits den Rahmen dar, in dem sich auch die betriebswirtschaftlichen Anforderungen bewegen. Einige Anforderungen stellen sich jedoch speziell in diesem Umfeld. Auf sie soll an dieser Stelle gesondert eingegangen werden.

2.2.3.1 Supply Chain Management

Die Bedingungen unter denen Unternehmen an ihren Märkten tätig sind, haben sich in den letzten Jahren grundlegend geändert. Durch die Globalisierung und den damit einhergehenden verschärften Konkurrenzdruck sind Unternehmen gezwungen, effektiver, kostengünstiger und dabei dennoch schneller und flexibler zu arbeiten [vgl. Corsten / Gabriel, 2004; Speyerer, 2005, 1]. Der Wandel von „Push“- zu „Pull“- Märkten verlangt eine weitere Öffnung der Unternehmen zu ihren Kunden hin [Ralev, 2004, 3], individualisierte Produkte und Servicemerkmale im Dienste des Kundennutzens werden immer mehr zu einem

entscheidenden Wettbewerbsfaktor [Corsten / Gabriel, 2004, 4]. Als ein geeignetes Instrument, diesen Anforderungen gerecht zu werden, wird das →*Supply Chain Management (SCM)* angesehen.

Eine prägnante Definition für den Begriff Supply Chain Management abzuleiten, fällt aufgrund der Komplexität und des Umfangs der zu berücksichtigenden Faktoren schwer. Die gemeinsame Basis der veschiedenen Definitionsansätze stellt jedoch die *ganzheitliche, unternehmensübergreifende Gestaltung von Lieferketten* dar [Ralev, 2004, 3]. Diese Basis findet sich in den Charakteristika wieder, die in der Literatur in unterschiedlicher Kombination zumeist die folgenden Punkte umfassen [Ralev, 2004, 3]:

- *Durchgängige Kundenorientierung*: Der Bedarf des Endkunden bildet den Ausgangspunkt und die zentrale Motivation aller Gestaltungsmaßnahmen und Unternehmensprozesse.
- *Geschäftsprozessorientierung*: Das SCM zielt auf eine optimale Gestaltung des Güter- und Informationsflusses sowie der disposiven Prozesse aller an der Supply Chain beteiligten Unternehmen.
- *Kooperative Zusammenarbeit*: Alle an der Supply Chain beteiligten Unternehmen beteiligen sich proaktiv am Informationsfluss.

In der Praxis umfasst SCM

> „... alle Tätigkeiten, die mit dem Durchsatz von Gütern vom Rohmaterialzustand bis hin zum Endverbraucher verbunden sind. Das schließt die Beschaffung, Konstruktion und Entwicklung, Produktionsplanung, Auftragsabwicklung, Bestandsmanagement, Transport, Lagern, Kundenservice und [...] das Informationssystem zur Kommunikation aller an der Supply Chain Beteiligten mit ein.“ [Wenzel u. a., 2001, 325]

Es steht außer Frage, dass eine solch weitreichende Zusammenarbeit zu ihrer Unterstützung auf mächtige Softwaresysteme zurückgreifen muss, die sowohl die einzelnen Teilsysteme der genannten Unternehmensprozesse integrative zusammenführen, als auch eine Ablaufsteuerung, sowie Monitoring- und Analyse-Werkzeuge bereitstellen müssen. Hier haben sich die großen Anbieter von Unternehmenssoftware, wie SAP, Oracle oder IBM, einen lukrativen und stetig wachsenden Markt geschaffen. Das Business Application Research Center (BARC) weist für das Jahr 2006 auf dem deutschen Markt für Business Intelligence Software einen Gesamtumsatz von 606 Millionen Euro aus. Tabelle 2.1 listet die fünf größten Anbieter auf dem deutschen Markt mit ihren Umsätzen in 2006.

Unternehmen	Umsatz in Mio. EUR	Anteil am Gesamtumsatz
SAS	87,0	13,7%
SAP	74,0	12,2%
Cognos	32,5	5,4%
Oracle	32,0	5,3%
IBM	31,5	5,2%

Tabelle 2.1: Umsatzanteile der fünf größen Anbieter von Business Intelligence Software in Deutschland im Jahr 2006. Quelle: Andreas Friedrich, http://www.barc.de/de/marktforschung/research-ergebnisse/top-10-softwareanbieter-im-deutschen-markt-fuer-business-intelligence.html, kostenloser Login erforderlich, letzter Aufruf am 07.02.2009

2.2.3.2 Legacy-Systeme

Als *Legacy-Systeme* (wörtlich übersetzt: Altlast-Systeme) werden Anwendungssysteme bezeichnet, deren Technologie nicht mehr den aktuellen Standards entspricht. Solche Systeme beruhen meist auf veralteten Programmiersprachen und Datenspeichertechnologien, die einen Datenaustausch durch fehlende Unterstützung erschweren oder verunmöglichen [vgl. Leser / Naumann, 2007, 250f]. Sie waren häufig über Jahre oder gar Jahrzehnte im Einsatz und in der Wartung, was einerseits zur Folge hat, dass vorgenommene Anpassungen kaum oder nicht dokumentiert sind oder die entsprechenden Administratoren nicht mehr Teil des Unternehmens sind. Andererseits sind die Daten dieser Systeme häufig von fundamentaler Bedeutung und das System wird aufgrund seiner speziellen Funktionalitäten so dringend benötigt, dass es permanent funktionsfähig sein muss. Deshalb sind Legacy-Systeme nur schwer oder überhaupt nicht zu ersetzen. Für die Integration solcher Systeme sind meist aufwendige Verfahren nötig, um die benötigten Daten verfügbar zu machen.
Aus betriebswirtschaftlicher Sicht muss bei der Integration solcher Systeme also ein optimaler Kompromiss gefunden werden, der mit vertretbarem Aufwand das System erhält, die Daten trotzdem nutzbar macht und das System für eingehende (von anderen Systemen ausgehende) integrative Operationen aufschließt.

2.3 Grundlegende Anwendungsfelder

Daten entstehen überall und bei jeder Aktivität in einem Unternehmen. Um diese Daten optimal für die betrieblichen Anwendungsfälle nutzen zu können, ist es nötig, sie über den gesamten Lebenszyklus hinweg angemessen zu verwalten und das Integrationspotential in jeder Phase zu erkennen und zu nutzen. Der Lebenszyklus der Daten reicht von der Erfassung und der Bearbeitung, über die Präsentation und Analyse, bis hin zur Archivierung und der Recherche. [Lohse, 2003, 19]

In [Lohse, 2003, 20-30] werden die folgenden grundlegenden Anwendungsfelder für die Integration abgeleitet, die in den konkreten Integrationslösungen auch kombiniert auftreten können:

Strukturübergreifende Präsentation: Sowohl innerhalb des Unternehmens, als auch in der Darstellung nach außen erfordern die große Konkurrenz und der steigende Wettbewerbsdruck eine möglichst umfassende, ansprechend aufbereitete und komfortabel zu bedienende Präsentation von Daten. Innerhalb des Unternehmens ist dieses Anwendungsfeld vor allem in der Bereitstellung von Unternehmensdaten für Manager und Entscheider zu nennen. Aber auch Mitarbeiter können von der Datenpräsentation einen Lern- und Nutzeffekt erfahren. Sie bekommen ein umfassenderes Bild ihres Unternehmens, die Identifikationsbereitschaft und das Verantwortungsbewusstsein können gestärkt werden. Unter der Präsentation nach außen finden sich sowohl Informationsmittel für potentielle Kunden und Partner, als auch teilweise der Markt selbst, über den das Unternehmen seine Leistungen oder Produkte anbietet.

Beispiele für das Anwendungsfeld der Präsentation sind u.a.:

- der elektronische Produktkatalog in einem Online-Shop
- die Unternehmenswebsite mit Kernzahlen zu Umsatz und Geschäftsverlauf, aufbereitet für Anleger, Partner und Kunden
- die grafische Darstellung der Entwicklung der Umsätze für ein Produkt zur Analyse durch das Management

strukturübergreifende Archivierung, Recherche und Analyse: Die anfallenden und bereits gespeicherten Daten müssen archiviert werden, um sie sicher vorzuhalten und bei Bedarf jederzeit wiederverwenden zu können. Dabei sind gesetzliche Rahmenbedingungen, etwa zum Datenschutz oder zur Vorhaltung von kundenbezogenen Daten zu beachten. Methoden zur Informationsgewinnung aus archivierten Daten dienen in erster Linie dem Management als Werkzeuge für strategische und analytische Prozesse der Unternehmensaufstellung

und sind damit ein entscheidender Wettbewerbsfaktor für das Unternehmen [Eulgem, 1998, 1]. Zu den Werkzeugen der Datenrecherche zählen die strukturübergreifende Datenanalyse, das Information Retrieval, sowie die Verdichtung von Informationen.

Erstellung, Änderung und automatische Bereitstellung von Daten: Ein hohes Optimierungspotential entsteht in Unternehmen auch dadurch, dass Mitarbeiter bei der Erfassung von Daten keine Kenntnis davon haben, ob dazu bereits an anderer Stelle im Unternehmen Daten komplett oder teilweise vorliegen. Die Integration soll hier Abhilfe schaffen, indem vor der Neuerfassung von Daten geprüft wird, ob diese oder partiell identische Daten bereits existieren, und indem den Mitarbeitern alle für ihre Tätigkeiten relevanten Daten komfortabel zur Verfügung gestellt werden. Für häufig wiederkehrende, gleichartige Änderungen der Daten können automatisierte Abläufe das manuelle Eingreifen eines Mitarbeiters auch gänzlich ersetzen.

Beispiele für die automatische Bereitstellung und Änderung von Daten sind:

- Patientenmappen für Ärzte
- das automatische Aktualisieren von Lagerbestandslisten durch Verkaufssysteme

Strukturübergreifende Kommunikation: Die Integration unterstützt die rechnergesteuerte Kommunikation zwischen Menschen direkt, aber auch den Austausch von Nachrichten zwischen den integrierten Systemen als folge einer Handlung eines menschlichen Benutzers des Integrationssystems.

2.4 Technische Problemdimensionen der Integration

Die folgenden Abschnitte stellen die technischen Kernproblemfelder vor, die bei der Erstellung von Integrationslösungen bearbeitet werden müssen. Sie treten vor allem bei der Informationsintegration in den Vordergrund, stellen sich jedoch auch bei der Anwendungsintegration. Diese Problemfelder sind die *Verteilung* der Daten, die *Autonomie* der Datenquellen, in denen die Daten vorliegen, sowie die *Heterogenität* zwischen den Datenquellen und der zu schaffenden Integrationslösung. Diese Problemfelder werden in [Leser / Naumann, 2007, 50] auch als *orthogonale Dimensionen der Informationsintegration* bezeichnet, was verdeutlichen soll, dass eine Kombination dieser Problemfelder in praktisch jeder Integrationslösung auftritt.

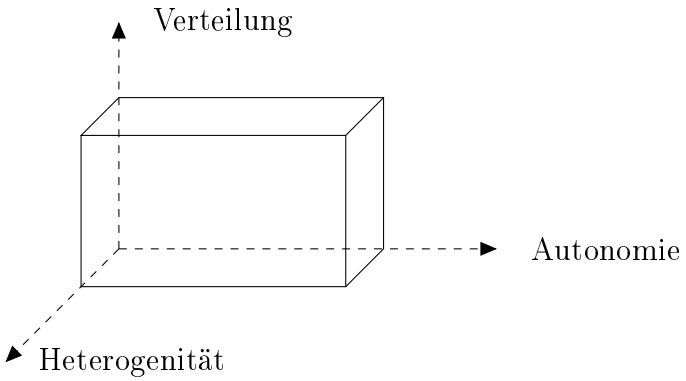

Abbildung 2.2: Technische Problemdimensionen der Integration. Erstellt nach [Leser / Naumann, 2007, 50].

2.4.1 Verteilung

Die *Verteilung von Daten* ist das offensichtlichste Problem bei der Integration. Daten werden von unterschiedlichen physischen Systemen erhoben und verarbeitet [Leser / Naumann, 2007, 51]. Beispielsweise befinden sich die Daten der Lagerhaltung eines Unternehmens in der Datenbank der Warenwirtschaftssoftware und die Kunden- und Bestelldaten in der Datenbank des E-Commerce-System. Das Integrationssystem muss diese Verteilung überwinden. Als Voraussetzung dafür muss angenommen werden, dass die verteilten Systeme untereinander, etwa über das Internet oder ein lokales Netzwerk, vernetzt sind. Sonst wäre ein Zugriff technisch unmöglich [Leser / Naumann, 2007, 51].

Die Verteilung von Daten hat nach [Leser / Naumann, 2007] zwei wesentliche Ausprägungen:

- *Physische Verteilung* liegt vor, wenn die Daten auf unterschiedlichen Systemen liegen, die damit auch räumlich voneinander entfernt sind. Physische Verteilung liegt auch dann vor, wenn die Daten auf zwei Servern liegen, die sich im selben Raum befinden, oder sie sich auf zwei verschiedenen → Virtual Machines auf demselben Rechner befinden.
- *Logische Verteilung* liegt dann vor, wenn es für ein Datum innerhalb des Systems mehrere mögliche Orte seiner Speicherung gibt. In diesem Fall entstehen Redundanzen im System, die das Integrationssystem widerspruchsfrei auflösen muss um ein konsistentes Gesamtbild zu erhalten.

Um die physische Verteilung zu überwinden, müssen die verteilten Systeme im Netzwerk identifizierbar und im laufenden Betrieb lokalisierbar sein [Leser / Naumann, 2007, 51]. Dies geschieht für das Integrationssystem in der Regel

transparent. Über einen → Uniform Resource Locator (URL) werden entfernte Systeme angesprochen. Deren Lokalisierung ist dann Aufgabe der Netzwerkebene eines Rechnersystems und fällt nicht in den Zuständigkeitsbereich des Integrationssystems [Leser / Naumann, 2007, 52].

Ein weiteres Problem, das sich aus der physischen Verteilung ergibt, ist das Problem der *Anfrageoptimierung*. Bei verteilten Datenquellen muss die Anzahl der Anfragen über das Netzwerk minimiert werden, da diese um Größenordungen mehr Zeit benötigen als lokale Anfragen. Darüber hinaus unterliegen Anfragen über das Netzwerk weiteren Beschränkungen. So ist etwa die Zeit zum Aufbau der Verbindung ein weiterer zu berücksichtigender Faktor, ebenso wie die beschränkte Bandbreite der Verbindung oder die anfallenden Kosten pro Verbindung. [Leser / Naumann, 2007, 52]

Wenn Daten mit gleicher Bedeutung an verschiedenen Orten im System gespeichert sind, liegt *logische Verteilung* und damit *Redundanz* vor [Leser / Naumann, 2007, 53]. Logische Verteilung ist bei der Informationsintegration fast immer gegeben, da sich die Teilsysteme in ihren benötigten Daten überlappen. Beispielsweise werden Adressdaten von Kunden sowohl in der Warenwirtschaftssoftware vorliegen, da diese den Versand steuert, als auch in der E-Commerce-Software, wo die Daten, zumindest bei Neukunden, erstmalig erfasst werden. Die Eliminierung der Redundanzen ist meist nicht möglich, da die einzelnen Anwendungssysteme unabhängig voneinander ihre Datenbestände verwalten und die entsprechenden Datenfelder für ihre Funktion erforderlich sind. Um bei redundanten Datenquellen ein konsistentes Gesamtbild zu erhalten, muss die Redundanz streng überwacht werden. Durch geeignete Maßnahmen zur Integritätssicherung muss erzwungen werden, dass an den verschidenen Speicherorten stets dieselben Daten vorliegen. [Leser / Naumann, 2007, 53]

Das Integrationssystem muss daher sicherstellen, dass:

- durch geeignete Lokalisationsverfahren alle relevanten Datenbestände in das Gesamtbild einfließen,
- doppelt vorhandene Datensätze entsprechend zusammengeführt werden (Duplikaterkennung),
- die Daten zu einem kosistenten und homogenen Gesamtbild aufbereitet werden,
- Datenmanipulationen durch das Integrationssystem in allen integrieren Datenquellen zum gleichen Ergebnis führen.

2.4.2 Autonomie

Autonomie bezeichnet die „Freiheit der Datenquellen, unabhängig über die von ihnen verwalteten Daten, deren Struktur und die Zugriffsmöglichkeiten zu entscheiden“ [Leser / Naumann, 2007, 54]. Die einzelnen Teilsysteme eines Integrationssystems sollen nach der Integration in gewohnter Weise autonom verwendet werden können. Ein Integrationssystem, welches bereits vorhandene Datenquellen integrieren soll, wird daher immer mit einer hohen Autonomie umgehen müssen [Leser / Naumann, 2007, 54].

Es können vier grundsätzliche Arten der Autonomie unterschieden werden:

Designautonomie bezeichnet die Freiheit einer Datenquelle zu bestimmen, in welcher Art und Weise sie ihre Daten zur Verfügung stellt. Dies umfasst das Datenschema, das Datenmodell, die syntaktische Darstellung der Daten, die Verwendung von Schlüssel- und Indexsystemen, sowie die Einheiten von Werten. [Leser / Naumann, 2007, 55] Durch die Designautonomie können schwierig aufzulösende Heterogenitäten in struktureller, syntaktischer und semantischer Hinsicht entstehen. Eine Möglichkeit, die durch Designautonomie entstehenden Konflikte zu mildern, ist die Definition von Austauschformaten oder Zugriffsschnittstellen, die dann die Daten der Datenquelle in definierter und möglichst leicht zu verarbeitender Form zur Verfügung stellen (vgl. Abschnitt **??**, S. **??**).

Schnittstellenautonomie bedeutet, dass die Datenquelle festlegt, mit welchen technischen Verfahren auf ihre Daten zugegriffen werden kann [Leser / Naumann, 2007, 55]. Beispielsweise kann sie den Zugriff über Webservices oder über Webformulare ermöglichen. Das Integrationssystem hat sich diesen technischen Möglichkeiten meist unterzuordnen.

Zugriffsautonomie liegt vor, wenn die Datenquelle selbst bestimmt, wer in welcher Weise auf welche ihrer Daten zugreifen kann [Leser / Naumann, 2007, 56]. Das umfasst sowohl die Benutzerauthentifizierung und Benutzerautorisierung, als auch die Vergabe von Lese- und Schreibrechten für bestimmte Daten. Auch hier hat das Integrationssystem meist keinen Einfluss und muss davon ausgehen, dass die Zugriffssteuerung korrekt funktioniert und konfiguriert ist.

Ausführungsautonomie bezeichnet die Unabhängigkeit der Datenquelle hinsichtlich der Ausführung von Datenbankoperationen [Höding, 2000, 15]. Die Datenquelle bestimmt selbst den Zeitpunkt, die Reihenfolge und den Umfang der Operationen. Dies betrifft neben der Abarbeitung der Anfragen auch die Verwaltung des Mehrbenutzerbetriebs mit Transaktionsverwaltung und Integritätssicherung.

2.4.3 Heterogenität

Als *heterogen* werden zwei Informationssysteme bezeichnet, wenn sie sich im Hinblick auf die angebotenen Modelle, Schemata und Methoden zum Zugriff auf die Daten nicht exakt gleichen [Leser / Naumann, 2007, 58]. Die Heterogenität von Datenquellen ist ein sehr weitreichendes Problemfeld und verursacht dem Unternehmen beträchtliche Kosten [Höding, 2000, 13]. Grundsätzlich ist festzustellen, dass der Grad der Heterogenität von Datenquellen mit dem Grad ihrer Autonomie zunimmt [Leser / Naumann, 2007, 58f]. Sie ergibt sich aufgrund unterschiedlicher Anforderungen an die Einzelsysteme und aufgrund der Tatsache, dass selbst identische Ausgangssysteme, die den selben unternehmnerischen Aufgaben dienen, fast immer Daten heterogen speichern.

Beispielhafte Gründe für das Auftreten von Heterogenität sind:

- Das Anpassen von eingekauften Ausgangssystemen an unternehmensspezifische Bedürfnisse
- Die Verwendung von speziellen Datenbanksystemen für verschiedene Teilsysteme im Unternehmen
- Die Notwendigkeit der Zusammenarbeit bisher getrennter Systeme aus verschiedenen Unternehmen, z.B. nach einer Unternehmensfusion

Heterogenität kann auf allen Ebenen der involvierten Teilsysteme auftreten. Neben der *Heterogenität auf Hardwareebene*, die durch den Entwicklungsstand der Vernetzung und Internetanbindung der Systeme heutzutage als weitgehend überwunden betrachtet werden kann [Höding, 2000, 15], werden im folgenden die *Heterogenität auf Datenmodellebene*, die *Heterogenität auf Datenschemaebene*, sowie die *Heterogenität auf Datenebene* unterschieden.

Heterogenität auf Datenmodellebene

Ein Datenmodell beschreibt die Struktur und die Bedeutung von Daten in einer Datenquelle [Heinrich u. a., 2004, 178]. Dazu wird eine Sammlung von Modellierungskonzepten verwendet [Höding, 2000, 16], die Verknüpfungen der Daten untereinander beschreibt. Man kann grob zwischen *entwurfsorientierten Modellierungskonzepten* und *implementierungsorientierten Modellierungskonzepten* unterscheiden [Höding, 2000, 16]. Die entwurfsorientierten Modellierungskonzepte stellen die Daten und deren Beziehungen unabhängig von der später zu verwendenden Datenhaltungstechnologie dar und dienen der Findung einer möglichst systematischen und anschaulichen Datenrepräsentierung. Das verbreitetste Werkzeug für die entwurfsorientierte Modellierung ist die → Unified

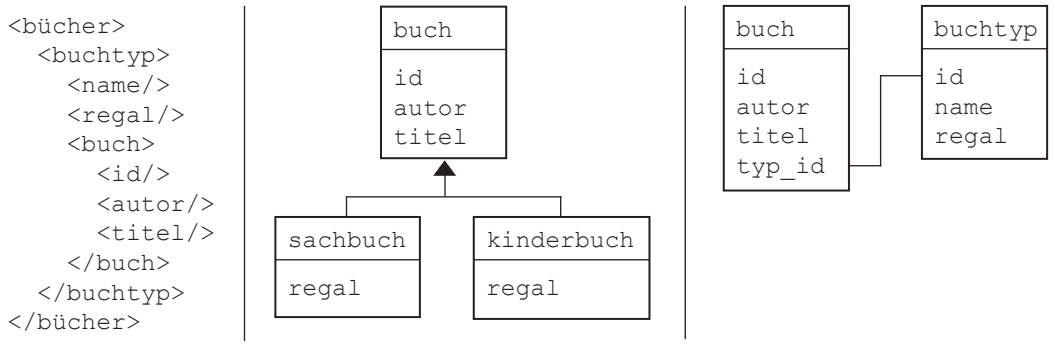

Abbildung 2.3: Heterogenität auf Datenmodellebene

Modelling Language (UML). Implementierungsorientierte Modellierungskonzepte bereiten Daten für die konkrete Implementierung in einer zu verwendenden Datenhaltungstechnologie auf. Sie definieren damit auch die technischen Schnittstellen zum Zugriff auf und Umgang mit den Daten und geben das Datenschema vor.

Heterogenität auf Datenmodellebene liegt vor, wenn sich das Integrationssystem und die Teilsysteme in ihrem Datenmodell, und damit in der Repräsention der Daten unterscheiden [Leser / Naumann, 2007, 65]. Abbildung 2.3 veranschaulicht die Datenmodellheterogenität. Jedes der drei Datenmodelle repräsentiert in etwa die gleichen Daten, verwendet jeoch unterschiedliche Modellierungskonzepte und unterschiedliche Datenhaltungssysteme und bietet daher auch unterschiedliche Schnittstellen zum Zugriff auf die Daten (Links: XML-Struktur; Mitte: Objekthierarchie; Rechts: relationale Datenbanktabellen). Die Verschiedenartigkeit der Datenmodelle erschwert die Vergleichbarkeit der Schemata, vor allem für das Integrationssystem, und damit die Integration [vgl. Höding, 2000, 17].

Heterogenität auf Datenschemaebene

Auch wenn zwei Datenquellen dasselbe Datenmodell für ihre Daten verwenden, können Daten – zumindest in semantisch reichen Datenmodellen – ohne Unterschiede in der Bedeutung auf mehrere Arten repräsentiert werden [vgl. Höding, 2000; Leser / Naumann, 2007]. Ist dies der Fall, so spricht man von *Datenschemaheterogenität*. Bspw. können in einem relationalen Datenmodell Eigenschaften ohne Bedeutungsänderung als Relationen, Attribute oder Attributwerte gespeichert werden (siehe folgendes Beispiel). Die Auswahl hat jedoch fundamentale Auswirkungen auf die Anfragemethoden, mit denen auf die Daten zugegriffen werden kann, sowie auf die Flexibilität des Schemas.

Das folgende Beispiel veranschaulicht die unterschiedlichen Modellierungsmöglichkeiten für die (bedeutungsgleiche) Unterscheidung verschiedener Buchtypen (`kinderbuch` bzw. `sachbuch`) in einem relationalen Datenmodell (in SQL-Notation):

- *Modellierung als Relation*:
  `sachbuch (id, autor, titel, seitenzahl)`
  `kinderbuch (id, autor, titel, seitenzahl)`
- *Modellierung als Attribut*:
  `buch (id, autor, titel, seitenzahl, sachbuch, kinderbuch)`
  wobei in den Attributen `sachbuch` und `kinderbuch` durch einen Integer-Wert von `1` (trifft zu), bzw. `0` (trifft nicht zu) festgehalten wird, welchem Typ das Buch angehört.
- *Modellierung als Attributwert*:
  `buch (id, autor, titel, seitenzahl, typ)`
  wobei im Attribut `typ` durch den String-Wert `kinderbuch`, bzw. `sachbuch` festgehalten wird, welche Wertausprägung die Eigenschaft einnimmt.

Ein erster Unterschied der Modellierungsvarianten wird deutlich, wenn man die Schritte betrachtet, die notwendig sind, um bspw. den Buchtyp `kochbuch` hinzuzufügen:

- Bei der *Modellierung als Relation* muss eine neue Relation erstellt werden.
- Bei der *Modellierung als Attribut* muss die Relation um ein Attribut erweitert werden. Bei solchen Erweiterungen ist immer gesondert auf die Einhaltung von Integritätsbedingungen zu achten [Leser / Naumann, 2007, 71].
- Bei der *Modellierung als Attributwert* muss das Schema nicht geändert werden. Es ist lediglich der Wertebereich des Attributs `buch.typ`, d.h. die Menge der Werte, die das Attribut einnehmen darf, zu erweitern.

Ein größeres Problem ergibt sich aus unterschiedlichen Modellierungsvarianten jedoch für die Datenabfragen des Integrationssystems. In der Regel können Abfragesprachen die Modellierungsunterschiede nicht überbrücken [Leser / Naumann, 2007, 71]. So müssen bspw. Attribute und Relationen immer explizit benannt werden. Ändern sie sich, müssen auch die entsprechenden Anfragen geändert werden. Möglich ist auch, dass Attribute, die dieselben Daten enthalten in unterschiedlichen Datenquellen verschieden heißen. So kann z.B. die Anzahl der Seiten eines Buches in der einen Datenquelle unter dem Attribut

`seitenzahl` und in einer anderen Quelle unter dem Attribut `num_pages` vorliegen.

Bei komplexen Tabellenabfragen können sich aus den verschiedenen Modellierungsmöglichkeiten signifikant verschiedene Performance-Werte bei der Verarbeitung ergeben. Es ist sogar möglich, dass spezielle Anfragen nicht von allen Modellierungsvarianten unterstützt werden können.

Heterogenität auf Datenebene

Die am schwierigsten aufzulösenden Heterogenitätsprobleme entstehen auf der Datenebene. Unter Datenheterogenität vesteht man „die Existenz unterschiedlicher Werte für dieselbe Eigenschaft eines in mehreren zu integrierenden Datenbanken abgespeicherten Realweltobjekts“ [Höding, 2000, 18]. Tritt Datenheterogenität auf, muss das Integrationssystem grundsätzlich einen der vorgefundenen Werte als korrekt für die weitere Verwendung identifizieren, bzw. den korrekten Wert herleiten. Dabei benötigt es spezifische Interpretationskenntnisse, denn es muss zuerst aus den Daten die enthaltenen Informationen extrahieren und gegeneinander auswerten, um die korrekten Informationen wieder als Daten zur Verfügung stellen zu können. Dies ist sehr aufwendig und mitunter sogar unmöglich.

Datentabelle A

id	name	gebdatum	geschl
24	Peter Müller	08.07.1967	m
25	Max Meier	12.07.1944	m
26	Frida Kalo	24.11.1981	f
27	Uwe Teichj	12.02.1976	m
28	Anja Bauer	05.04.1973	f

Datentabelle B

id	name	gebdatum	geschl
24	Becker, K.	02.SEP	Männl.
25	Meier, M.	12.JUL	Männl.
26	Kahlo, F.	24.NOV	Weibl.
27	Teich, U.	12.FEB	Männl.
28	Bauer, A.	14.SEP	Weibl.

Abbildung 2.4: Heterogenität auf Datenebene

Die Abbildung 2.4 veranschaulicht beispielhaft einige Ausprägungen von Datenheterogenität. Es sollen zwei Datentabellen mit Kundendaten integriert werden (Datentabelle A und Datentabelle B). Die Datensätze (also die Zeilen der Tabellen) stellen dabei idealerweise dasselbe Datenobjekt, also denselben Kunden dar. Weiterhin wird angenommen, dass die `id` einen Kunden eindeutig referenziert, dass also der Datensatz mit der `id` 123 in beiden Systemen dieselbe Person repräsentiert. Die Datensätze der Tabellen weisen dann folgende Heterogenitäten auf:

- Das Geschlecht wird unterschiedlich kodiert. Für die Integration müssen die verschiedenen Kodierungen ineinander überführt werden können. Die

Information stimmt jedoch überein, d.h. beide Kodierungen enthalten die gleiche Information.

- Das Geburtsdatum wird unterschiedlich kodiert. In Datentabelle B wird außerdem das Geburtsjahr nicht mit geführt. Diese Information stellt nur Datentabelle A bereit. Bei der Person mit der `id` 28 weichen die Angaben zum Geburtsdatum voneinander ab. Es ist nicht entscheidbar, welches Datum zutrifft und welches nicht.
- Die Namen der Kunden werden unterschiedlich kodiert. Der volle Vorname ist nur in Datentabelle A vorhanden. Zusätzlich treten Abweichungen bei den Namen auf. Während bspw. bei der Person mit der `id` 27 die Ursache wahrscheinlich ein Tippfehler ist, und daher entschieden werden kann, welcher Name korrekt ist, kann dies bei der Person mit der `id` 26 nicht so leicht hergeleitet werden.
- Bei der Person mit der `id` 24 weichen sowohl der Name, als auch das Geburtsdatum voneinander ab. Hier ist weder entscheidbar, ob einer der beiden Datensätze zutrifft, noch ob beide korrekt oder falsch sind.

2.4.4 Weitere Problemfelder

Neben den geschilderten Problemdimensionen treten noch weitere Schwierigkeiten auf, die bei der Integration beachtet werden sollten:

- Die vielfältigen Wechselwirkungen integrierter Systeme machen Programmänderungen und Programmtests sehr kompliziert [Mertens / Griese, 1993, 10].
- Fehler breiten sich in integrierten Systemen viel leichter aus und können unter Umständen schwer zu beseitigen sein und große Teile des Gesamtsystems lahmlegen. Je mehr Teilsysteme in einem integrierten System korrekt zusammenarbeiten müssen, desto geringer ist die Wahrscheinlichkeit, dass das Gesamtsystem fehlerfrei funktioniert. Daher sollten beim Design eines Integrationssystems immer auch sog. „Brandmauern“, also Absicherungen des Datenbestandes, bspw. durch das Vorhalten von Zwischen- und Teilbeständen, realisiert werden, um das System im Fehlerfall schneller und modularisiert korrigieren zu können. [vgl. Keller, 2002, 177ff]

2.5 Kategorisierung der Integrationsansätze

Integration kann nach verschiedenen Kriterien kategorisiert werden, welche in Abbildung 2.5 überblicksartig zusammen gefasst sind. Auf einige Ausprägungen wird im folgenden detaillierter eingegangen.

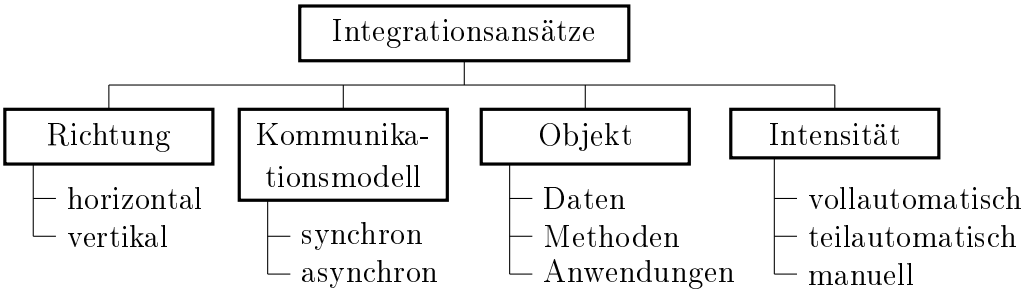

Abbildung 2.5: Kategorisierung der Integrationsansätze

2.5.1 nach Richtung

2.5.1.1 Horizontale Integration

Im Kontext einer SCM-Integration bezeichnet horizontale Integration die „Verknüpfung von Teilbereichen betrieblicher Prozesse entlang der [...] Wertschöpfungskette (Supply Chain)“ [Schwarze, 2000, 133]. So ist z.B. ein System, das die Abwicklung von Aufträgen von der Materialbeschaffung bei externen Firmen, über die Fertigung bis hin zur Auslieferung regelt, ein Vertreter der horizontalen Integration. Horizontale Integration dient damit vorwiegend der Unterstützung von Geschäftsprozessen [Heine, 1999, 79].

2.5.1.2 Vertikale Integration

Die vertikale Integration bezeichnet die Zusammenfassung von Abläufen derselben Teilbereiche auf verschiedenen Supply-Chains, bzw. die Integration von Managementaufgaben. Ein Beispiel für ein solches System ist eine Anwendung, welche die Einkaufsvorgänge von verschiedenen Zulieferern für verschiedene Produkte vergleichen kann. Die vertikale Integration dient somit in erster Linie zur informationstechnischen Unterstützung des Management [Heine, 1999, 76].

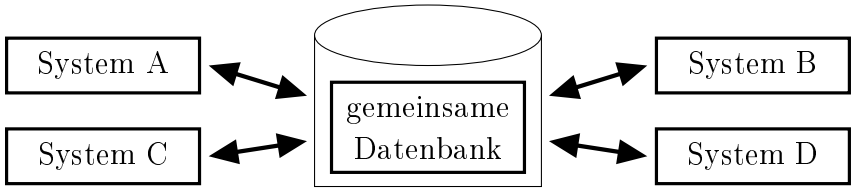

Abbildung 2.6: Schema der Informationsintegration [Schwarze, 2000, 134]

2.5.2 nach Objekt

2.5.2.1 Informationsintegration

Die *Informationsintegration* (auch Datenintegration genannt) ist „eine Form der organisatorischen Integration, bei der die Datenbasis so organisiert wird, dass unterschiedliche Funktionen die gleichen Daten verwenden, also die Daten ohne Redundanz geführt werden“ [Heinrich u. a., 2004, 175].

Informationsintegration kann grundsätzlich in zwei Ausprägungen vorliegen. Bei der *virtuellen Informationsintegration* (dynamischen Integration) wird die Integration erst zum Zeitpunkt des Zugriffs realisiert, bspw. durch das Anfragen an mehrere Datenquellen und die Darstellung eines integrierten Ergebnisses. Die Daten werden für das Ergebnis aus den Datenquellen „gezogen“, weshalb man auch vom *Pull-Modus* spricht, und sofort nach Beendigung der Anfrage wieder verworfen [Leser / Naumann, 2007, 86]. Die Vorteile der virtuellen Integration liegen in der großen Flexibilität und der Aktualität der Daten [Leser / Naumann, 2007, 88]. Die einzelnen Datenquellen bestehen autonom weiter. Dort müssen keine Datentransformationen vorgenommen werden, und die Systeme, welche einzelne Datenquellen direkt verwenden, müssen nicht angepasst werden. Da die Daten erst zum Zeitpunkt der Anfrage integriert werden, sind sie immer auf dem aktuellsten Stand. Dies wird auch als *synchrone Bereitstellung* bezeichnet.

Dafür ist der Aufwand bei der Verarbeitung und der dynamischen Erstellung des Ergebnisses größer. Die Abfrage muss in Teilabfragen zerlegt werden, die über das Internet abgesetzt, und von den jeweiligen Datenquellen beantwortet werden. Anschließend müssen die Einzelergebnisse vereinigt und transformiert werden. Dies kostet Zeit und Rechenleistung. [Leser / Naumann, 2007, 88] Auch die Ausführung von Transaktionen kann sich problematisch gestalten, da aus Gründen der Datenkonsistenz sichergestellt werden muss, dass die an einer Abfrage beteiligten Datenoperationen auf allen Datenquellen erfolgreich ablaufen oder aber komplett rückgängig gemacht werden können.

Im Gegensatz dazu steht die *materielle Informationsintegration* (statische Integration). Hier liegen die integrierten Daten zum Zeitpunkt des Zugriffs bereits fertig und systemintern, also im lokalen Anwendungsportfolio, vor. Die einzelnen Datenquellen existieren weiterhin, werden aber für die Arbeit mit dem integrierten System nicht weiter verwendet. [Leser / Naumann, 2007, 86] Der Vorteil liegt hier in der konsistenten und sehr redundanzarmen Speicherung der Daten, sowie in den optimalen Zugriffsbedingungen. Es können bei der materiellen Integration alle gebotenen Funktionalitäten des verwendeten Datenbanksystems verwendet werden, bspw. die Verarbeitung von komplexen Anfragen über mehrere Tabellen, oder das Transaktionsmanagement zur Konsistenzsicherung.

Problematisch ist bei der materiellen Integration die Aktualität der Daten. Diese werden meist asynchron bereitgestellt, d.h. sie werden in definierten Intervallen von den einzelnen Datenquellen auf das Integrationssystem „geschoben“ – man spricht hier vom *Push-Modus*. Die Aktualität der Daten hängt damit von der Frequenz der Aktualisierung ab. [Leser / Naumann, 2007, 88] Dies macht die materielle Integration für bestimmte Anwendungsgebiete, in denen es auf Echtzeit-Daten ankommt, unbrauchbar. Auch der Speicherbedarf ist bei einer materiellen Integration naturgemäß hoch. In großen Unternehmen erreichen Datenbanken nicht selten Größen im Terabyte-Bereich [Lohse, 2003, 23]. Die Einzelhandelskette Wal-Mart, um nur ein Beispiel zu nennen, führt Daten über bis zu 200.000 Produkte und allein über die in den Filialen eingesetzten Scannerkassensysteme fallen täglich mehrere Gigabyte an Daten an [Wittmann u. a., 2000, 17]. Diese Angaben stammen wohlgemerkt aus dem Jahr 2000.

Für den Anwender ist die Unterteilung in virtuelle und materielle Integration transparent, d.h. nicht sichtbar, und daher nicht relevant [Leser / Naumann, 2007, 8]. Zumindest wenn das Integrationsintervall klein genug gehalten wird. Wenn aber z.B. die Nachrichten eines integrierten Web-Portals irgendwann 48 Stunden alt sind, weil seit diesem Zeitpunkt keine erfolgreiche Integrationsoperation mehr stattfand, dann merkt der Nutzer das irgendwann doch.

2.5.2.2 Anwendungsintegration

Die *Anwendungsintegration* (auch als Programmintegration bezeichnet) ist die Integration über Programmabläufe und Programmverknüpfungen [Schwarze, 2000, 134]. Daten werden von einem Teilsystem erfasst, bzw. erstellt und dann über einen Zwischenspeicher an andere Teilsysteme zur Verarbeitung weitergegeben [Lohse, 2003, 11]. Das Ziel der Anwendungsintegration ist es, die gewach-

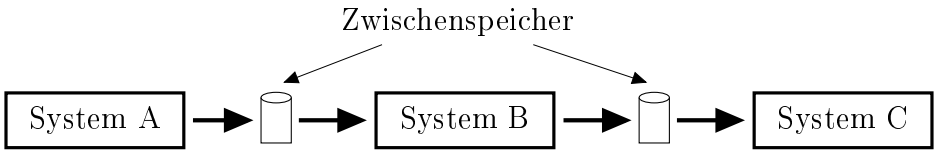

Abbildung 2.7: Schema der Anwendungsintegration [Schwarze, 2000, 134]

senen, „heterogenen Anwendungen eines Unternehmens so zu integrieren, dass sie sich verhalten, als wären sie von Anfang an dafür entworfen worden, die aktuellen Geschäftsprozesse eines Unternehmens zu unterstützen“ [Keller, 2002, 5]. Der Begriff Anwendungsintegration wird gelegentlich gleichbedeutend mit →Enterprise Application Integration (EAI) verwendet [z.B. in Heinrich04, 63]. Der Terminus EAI bezieht sich aber ausschließlich auf die Integration innerbetrieblicher Teilsysteme, während Anwendungsintegration auch für die überbetriebliche Vernetzung von Anwendungssystemen Verwendung finden kann [Speyerer, 2005, 9].

Bei der Anwendungsintegration übernimmt jeweils ein Teilsystem die führende Rolle für einen bestimmten Vorgang [Lohse, 2003, 12]. In diesem System wird der Vorgang bearbeitet und resultierende Operationen in anderen Teilsystemen werden vom führenden System ausgelöst. Für verschiedene Vorgänge kann das führende System unterschiedlich sein, abhängig davon, welches System für die Bearbeitung des Vorgangs verwendet wird, bzw. am besten geeignet ist.

2.5.3 nach Kommunikationsmodell

Bei den verschiedenen Integrationsmethoden (siehe Abschnitt 2.6, S. 30ff) ist eine Kommunikation der einzelnen Teilsysteme, entweder untereinander (z.B. bei der nachrichtenbasierten Kommunikation, Abschnitt 2.6.2, S. 34) oder mit einer vermittelnden Instanz (z.B. bei der Integration über föderierte Datenbanken, Abschnitt 2.7.3, S. 49) nötig. Diese Kommunikation kann entweder *synchron* oder *asynchron* abgewickelt werden.

2.5.3.1 Synchroner Ansatz

Bei der Anwendung des synchronen Kommunikationsansatzes, der auch als *Request/Reply-Stil* bezeichnet wird, verharrt der Sender nach dem Absenden einer Nachricht solange (wird blockiert), bis eine Antwort des Empfängers eintrifft. Erst dann setzt der Sender seine Arbeit fort. [Keller, 2002, 77] Dies ist dann anwendbar, wenn

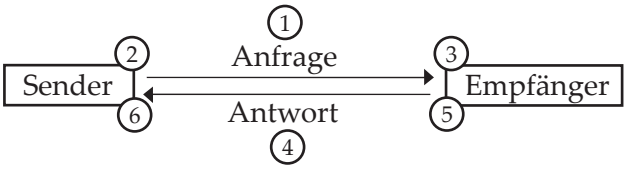

Abbildung 2.8: Ablauf synchroner Kommunikation: 1) Sender sendet Anfrage und 2) wird bis zur Antwort blockiert. 3) Empfänger empfängt und verarbeitet die Anfrage, 4) sendet die Anwort und 5) arbeitet danach weiter. 6) Nach Empfang der Antwort arbeitet auch der Empfänger weiter. Erstellt nach [Keller, 2002, 76f]

- der Sender ohne das Ergebnis der Anfrage nicht weiterarbeiten kann,
- sichergestellt ist, dass eine Antwort eintreffen wird, der Empfänger also zum Sendezeitpunkt arbeitet und die Nachricht ihn erreicht,
- die Anbindung des Empfängers an den Sender eine geringe Latenzzeit aufweist, bspw. wenn beide sich in einem lokalen Netzwerk befinden.

Der synchrone Kommunikationsansatz liegt auch den entfernten Funktionsaufrufen, → Remote Procedure Call (RPC) und darauf aufbauenden Infrastrukturen, wie → Distributed Computing Environment (DCE), zugrunde [Keller, 2002, 77]. Er findet vor allem dann Anwendung, wenn die Systeme im Betrieb eine untrennbare Einheit bilden und immer gemeinsam einsatzbereit sind, wie etwa die Anwendungssysteme und der Föderierungsdienst bei der Integration über föderierte Datenbanken. Außerdem gestaltet sich die Fehlerbehandlung auf Senderseite sehr einfach, da nahezu sofort ersichtlich ist, ob das Senden und Beantworten der Nachricht korrekt verlaufen ist [Keller, 2002, 79].

2.5.3.2 Asynchroner Ansatz

Wird der asynchrone Kommunikationsansatz, das sog. *Message Passing*, verwendet, so arbeitet der Sender nach dem Abschicken einer Nachricht sofort weiter, ohne auf die Antwort zu warten [Keller, 2002, 77]. Man nennt diesen Ansatz auch „fire and forget“, da der Sender praktisch „vergisst“, dass er eine Nachricht abgesetzt hat. Der Empfänger verarbeitet die Nachricht und sendet die Antwort seinerseits asynchron. So wird der Sender der ursprünglichen Nachricht zum Empfänger der Antwort, die er asynchron verarbeitet, wenn er dazu benötigte Ressourcen zur Verfügung hat. Sender und Empfänger sind also weitgehend entkoppelt. Anwendungsfälle für die asynchrone Kommunikation sind daher u.a.:

- die Kommunikation zwischen entfernten Systemen, deren momentane Verfügbarkeit nicht garantiert ist,
- wenn der Sender bis zum Eintreffen der Antwort noch weitere Operationen durchführen kann,
- wenn der Empfänger nicht bekannt ist, z.B. in einem EAI-System, das über einen Message Broker arbeitet.

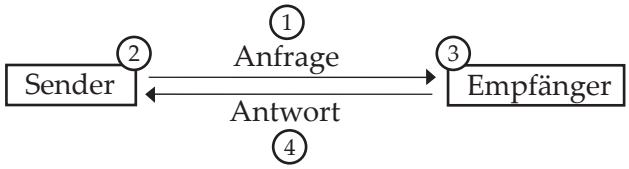

Abbildung 2.9: Ablauf asynchroner Kommunikation: 1)Sender sendet Anfrage und 2) arbeitet normal weiter. 3) Empfänger empfängt und verarbeitet Anfrage, 4) sendet die Antwort und arbeitet weiter. Erstellt nach [Keller, 2002, 77]

Vorteilhaft ist beim Message Passing auch, dass Nachrichten selbst Objekte sind, die für bestimmte Zwecke speicherbar sind. So können Nachrichten etwa archiviert, transformiert oder erneut verwendet werden [Leser / Naumann, 2007, 402]. Problematisch dagegen ist, dass der Sender keine Informationen darüber erhalten kann, ob und wann der Empfänger die Nachricht bearbeitet [Keller, 2002, 79]. Er muss sich darauf verlassen, dass die Antwort irgendwann eintreffen wird.

2.6 Kategorisierung von Integrationsmethoden

Heute eingesetzte objektorientierte Anwendungssoftware ist intern in der Regel in verschiedene Schichten aufgeteilt, die unterschiedliche Aufgaben der Anwendung getrennt voneinander bereitstellen. Ein grundlegendes Architektur-Paradigma ist das *Model View Controller -Paradigma* (MVC). Dabei werden das Model-Objekt, das das eigentliche Anwendungsobjekt darstellt, das View-Objekt, welches für die Bildschirmrepräsentation zuständig ist, und das Controller-Objekt, das die Interaktionsmöglichkeiten zwischen Anwendung und Benutzer bestimmt, unterschieden [Gamma u. a., 2003, 5]. Diese Objekte sind voneinander unabhängig und nur lose gekoppelt. Sie können voneinander unabhängig verändert, ergänzt oder ausgetauscht werden. Dies geschieht vor

allem, um die Anwendung leichter und flexibler weiterentwickeln und warten zu können [Gamma u. a., 2003, 5]. Aus der Model-Ebene können einfach wartbare und flexible funktionale Schnittstellen zur Verfügung gestellt werden, die Datenobjekte direkt ein- und auslesen können.

Gerade in älteren Systemen ist dies aber nicht immer der Fall. Bis in die 1980er Jahre hinein wurden Anwendungen oft als monolithische Strukturen implementiert, in denen Anwendungs-, Präsentations- und Datenlogik vermischt wurden [Keller, 2002, 183]. Bei älteren Systemen können Daten auch in proprietäten Dateisystemen gespeichert sein, auf die ein Datenzugriff von außen nicht möglich ist. Ein weiteres Hindernis bei der Integration alter Systeme ist, dass im Laufe der Verwendung Datenfelder oftmals mit Inhalten überschrieben werden, die dort gar nicht vorgesehen waren [Keller, 2002, 183f]. So ist es z.B. denkbar, dass in einer Anwendung, die bei ihrer Entwicklung noch keine Email-Adressen erfassen musste, weil dieses Kommunikationsform noch nicht verwendet wurde, ein anderes Feld, wie „sonstiges“ oder „reserve4“ genutzt wird.

Gerade bei Softwaresystemen, die die Abläufe von ganzen Unternehmen über längere Zeiträume hinweg unterstützen, ist die Komplexität oftmals so groß, dass Änderungen am System ausgesprochen schwierig und teuer sind. In [Thome, 2006, 148f] wird die These vertreten, dass für solch komplexe Anwendungssysteme die Bezeichnung „Software“ geradezu gefährlich geworden sei, da sie völlig falsche Assoziationen zu ihrer Komplexität und Anpassungsfähigkeit auslöse. Thome schlägt für solche Anwendungen den Begriff „Complexware“ vor, der einen größeren mentalen Respekt auslöse und die Gefahren und Schwierigkeiten stärker herausstelle.

Obwohl eine Schichtenteilung in der Praxis also oftmals wesentlich komplexer oder für ältere Systeme nicht in der Form vorhanden ist, können drei grundlegende Softwareschichten aufgeführt werden, in denen sich die Integration vollziehen kann [vgl. Keller, 2002, 60f; Speyerer, 2005, 11]:

1. Die *Präsentationsschicht* stellt die Benutzungsschnittstelle bereit, sorgt also für die Datenannahme und die Datenausgabe. Außerdem wird die Programmablaufsteuerung der Präsentationsschicht zugerechnet.
2. In der *Anwendungsschicht* ist die eigentliche Geschäftslogik angesiedelt. Hier werden Daten gemäß der implementierten Funktionalitäten transformiert und verarbeitet und alle Operationen und Berechnungen durchgeführt.
3. Die *Datenhaltungsschicht* dient der Verwaltung und dem Zugriff auf die persistenten Daten der Anwendung. Häufig kommen hier relationale Da-

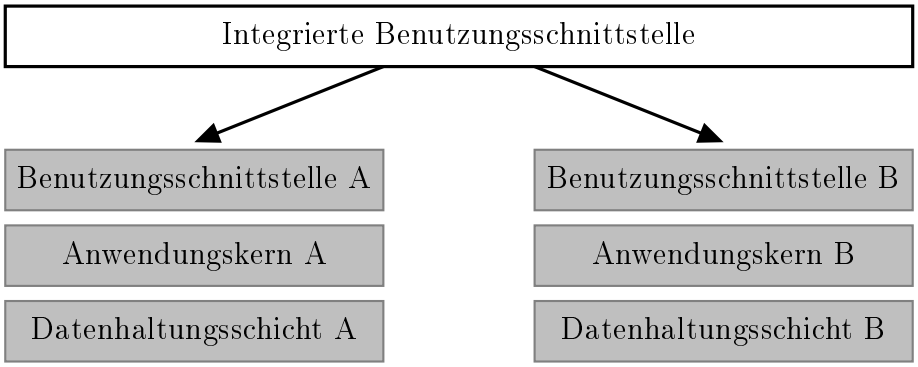

Abbildung 2.10: Integration über die Präsentationsschicht. Erstellt nach [Keller, 2002, 61].

tenbanken zum Einsatz, deren Managementsysteme weitere Funktionalitäten, z.B. zur Integritätssicherung, bereitstellen.

2.6.1 Integration über die Präsentationsschicht

Die Grundidee der Integration über die Präsentationsschicht besteht darin, „auf die Benutzungsschnittstellen existierender Anwendungen eine neue Benutzungsschnittstelle aufzusetzen“ [Keller, 2002, 61]. Das Hauptziel einer solchen Integrationslösung liegt entweder in einer besseren Benutzbarkeit der neuen Benutzungsschnittstelle, oder der besseren Abbildung von Geschäftsprozessen durch die integrierte Ansicht auf mehrere der zugrunde liegenden Schnittstellen [Keller, 2002, 61]. In der praktischen Realisierung kommt vor allem die Methode des → Screen Scraping zum Einsatz. Die Eingabefelder, Navigationsstrukturen und Datenausgaben werden dabei durch eine Software angesteuert und automatisch bedient, die eine Benutzung durch einen menschlichen Nutzer simuliert und die Ergebnisse dieser Aktionen über die neue Benutzungsschnittstelle ausgibt.

Bei der Verwendung einer solchen Integrationslösung sollten die folgenden Problemaspekte berücksichtigt werden:

- Eine solche Integrationslösung kann immer nur die Funktionalitäten besitzen, welche die zugrunde liegenden Benutzungsschnittsellen bereitstellen [Speyerer, 2005, 13]. Die Realisierung neuer Funktionen ist mit dieser Integrationsmethode allein nicht möglich.
- Bereits geringfügige Änderungen an den zugrunde liegenden Benutzungsschnittstellen, wie etwa das Einfügen oder Umbenennen eines Attributs, bedingen auch Veränderungen an der Integrationslösung.

- Das Auslesen und automatische Bedienen von Benutzungsschnittstellen ist keineswegs trivial. Zudem muss das Integrationssystem die Daten, mit denen es umgeht, angemessen verwalten können [Keller, 2002, 65]. Allerdings gibt es hierfür am Markt eine große Anzahl von fertig einsetzbaren Lösungen.
- Häufig stellt eine Anwendung nicht alle benötigten Daten über ihre Benutzungsschnittstellen zur Verfügung [Speyerer, 2005, 13]. Oder aber es sind mehrere Schritte nötig, um einen Vorgang zu bearbeiten, bspw. das Ausfüllen mehrerer Formulare in unterschiedlichen Programmteilen (Kundendaten, Bestelldaten) für die Aufnahme einer Bestellung [vgl. Keller, 2002, 184].

Die wesentlichen Anwendungsfelder für die Integration über die Präsentationsschicht liegen zum einen in der *Integration von Legacy-Systemen* (siehe Abschnitt 2.2.3.2, S. 14). Diese Systeme bieten oftmals keine andere Schnittstelle zur Integration an. Zum anderen wird die Methode bei der *webbasierten Integration durch Portale* (siehe Abschnitt 2.7.2, S. 48) benutzt, um mehrere internetverfügbare Anwendungen zu integrieren.

2.6.2 Integration über die Anwendungsschicht

Bei der Integration auf der Anwendungsschicht werden aus der Anwendungsschicht eines Teilsystems Funktionalitäten bzw. Daten direkt aus der Anwendungsschicht eines anderen Teilsystems aufgerufen [Keller, 2002, 66]. Voraussetzung dafür ist allerdings, dass die Anwendungskerne der Teilsysteme über Schnittstellen, → Application Programming Interface (API) genannt, Zugriff auf ihre Funktionalitäten erlauben [Speyerer, 2005, 13]. Nur dann ist diese Methode anwendbar. Sie hat folgende Vorteile:

- Bestehende Funktionalitäten können von allen integrierten Teilsystemen verwendet werden, ohne dass diese neu implementiert werden müssen.
- Neue Funktionalitäten müssen nur an einer Stelle implementiert werden, und zwar genau dort, wo die Funktionalität und die benötigten Daten strukturell und logisch hingehören.
- Die Anwendungskerne können in ihren Funktionalitäten selbst sicherstellen, dass die integrativ weiterzugebenden Daten die optimale Qualität, Validität und Relevanz haben.

Wie der Zugriff auf die Methoden im Anwendungskern genau erfolgt, hängt von einer Reihe von Faktoren und Designentscheidungen ab. So spielt z. B.

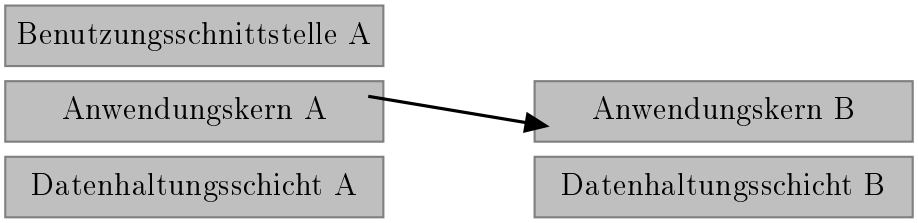

Abbildung 2.11: Integration über die Anwendungsschicht. Erstellt nach [Keller, 2002, 66].

die Verteilung (siehe Abschnitt 2.4.1, S. 17) eine Rolle, denn es macht einen Unterschied, ob die aufgerufene Funktion auf dem selben Rechner im gleichen →Adressraum oder auf einem entfernten Rechner läuft [Keller, 2002, 66].
Grundsätzlich können zwei Methoden der Integration über die Anwendungsschicht unterschieden werden: die Integration über *interfacebasierte Kommunikation* einerseits, und über *nachrichtenbasierte Kommunikation* andererseits.

Interfacebasierte Kommunikation

Bei der interfacebasierten Kommunikation kommt der synchrone Kommunkationsansatz zwischen aufgerufenem und aufrufendem System zum Einsatz (vlg. Abschnitt 2.5.3.1, S. 28). Man spricht dann von →Remote Procedure Call (RPC) [Leser / Naumann, 2007, 402]. Bei Anwendungen, die sich innerhalb desselben →Adressraums, oder in einer geeigneten Umgebung (z.B. →Distributed Computing Environment (DCE)) befinden, ruft eine Funktion in der Anwendungsschicht A eine Methode aus der Anwendungsschicht B eines anderen Systems direkt auf (siehe Tabelle 2.2). Der Vorteil der bei der Nutzung von Schnittstellen liegt in der *Typsicherheit* [Keller, 2002, 72]. Beim →Kompilieren, bzw. →Interpretieren wird geprüft, ob alle verwendeten Methoden und Argumente die richtigen Datentypen liefern. Fehler in der Typisierung liefern sofort eine Fehlermeldung. Somit wird sichergestellt, dass die Funktionalitäten korrekt zusammenarbeiten und die richtigen Daten und Argumente verwendet werden.

Nachrichtenbasierte Kommunikation

Bei der nachrichtenbasierten Kommunikation, dem sog. *Message Passing*, sind die Teilsysteme stärker voneinander entkoppelt [Keller, 2002, 75]. Der Sender

```
1
2 ...
3
4 // Aufgerufener Code (Anwendungssystem B):
5 public function doSomething(withThat, andWithThat) {
6   // do something here
7   return something;
8 }
9
10 ...
11
12 // Aufrufender Code (Anwendungssystem A):
13 include "doSomething";
14 var a=2, b=4;
15 var result = doSomething(a, b);
16
17 ...
```

Tabelle 2.2: Aufruf einer Funktion in Anwendungskern B durch Anwendung A

verschickt eine Nachricht, in welcher der Empfänger, die angeforderte Funktionalität und ggf. benötigte Argumente enthalten sind. Diese Nachricht wird über ein Transportmedium (Message Queue) an den Empfänger übermittelt, der aufgrund der Nachricht entscheidet, ob und wie er auf diese Nachricht reagiert. Der Ablauf wird in Abbildung2.12 veranschaulicht. Beim → Kompilieren, bzw. → Interpretieren müssen Sender und Empfänger dann nur noch die Schnittstelle kennen, die es ihnen erlaubt, Nachrichten zu senden und zu empfangen. Die Nachrichten selbst unterliegen nur geringen formalen Anforderungen [Keller, 2002, 75]. Am verbreitesten sind XML-Dokumente. Sender und Empfänger müssen in ihren Funktionalitäten selbst implementieren, wie fehlerhafte oder ausbleibende Nachrichten behandelt werden sollen.

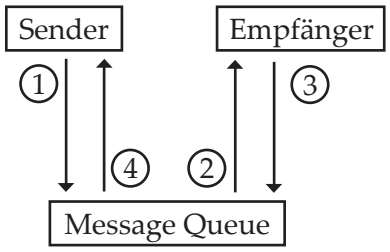

Abbildung 2.12: Kommunikation durch Message Passing. 1) Sender sendet Nachricht asynchron 2) Message Queue verarbeitet Nachricht und leitet sie an Empfänger weiter 3) Empfänger verarbeitet Nachricht und sendet Antwort asynchron 4) Message Queue verarbeitet Nachricht und leitet sie an Sender zurück. Erstellt nach [Keller, 2002, 74]

2.6.3 Integration über die Datenhaltungsschicht

Das Prinzip der Integration über die Datenhaltungsschicht ist, die Teilsysteme über eine gemeinsame Datenbasis kommunizieren zu lassen [Keller, 2002, 67]. Da die Datenhaltungsschicht für fast alle Funktionalitäten die Basis liefern muss, sind Änderungen an dieser automatisch für alle integrierten Teilsysteme in gleicher Weise verfügbar. Die Informationsintegration (vgl. Abschnitt 2.5.2.1, S. 26) baut nahezu ausschließlich auf dieser Möglichkeit auf. In der Praxis bringt dieser Ansatz jedoch auch einige Probleme mit sich:

- Die Anwendungsschicht der Teilsysteme wird komplett umgangen. Dadurch kann es nötig werden, gleiche Funktionalitäten in mehreren Teilsystemen einzeln zu implementieren [Speyerer, 2005, S.15].
- Das Datenbankmanagementsystem muss sicherstellen, dass die Integrität der Datenbasis bei allen Operationen von Teilsystemen erhalten bleibt. Dies schließt vor allem auch Transaktionsmanagement, Mehrbenutzersteuerung und das temporäre Sperren von in Bearbeitung befindlichen Datensätzen ein.
- Die Teilsysteme müssen für die Nutzung der integrierten Datenbasis angepasst werden. Die Integration über die Datenhaltungsschicht bringt häufig auch Änderungen im Datenschema mit sich, die in die Abfragen der Teilsysteme eingearbeitet werden müssen. Dies kann je nach Architektur der Teilsysteme sehr aufwändig sein.

2.6.3.1 Drei-Schichten-Architektur von Datenbanksystemen

Die grundlegende Architektur von Datenbanksystemen (DBMS), die sog. *Drei-Schichten-Architektur* [vgl. Leser / Naumann, 2007, 84f], wird in Abbildung 2.13 dargestellt. Die drei verschiedenen Schichten bieten jeweils eine andere *Sicht* durch ein eigenes Schema auf die gespeicherten Daten, und gewährleisten so die *Datenunabhängigkeit*.

Die *interne Sicht* verwaltet das physikalische Speichern der Daten. Sie wird auch als „physische Sicht“ bezeichnet. Das interne Schema definiert, welche Daten auf welchem Medium (z.B. Festplatte, Magnetband) an welcher Speicheradresse (Zylinder, Block) gespeichert werden.

Die *konzeptionelle Sicht* (auch „logische Sicht“) beinhaltet die konzeptionelle Modellierung der Daten. Im zugehörigen *konzeptionellen Schema* wird definiert, welches Datenmodell verwendet wird, welche Daten gespeichert werden,

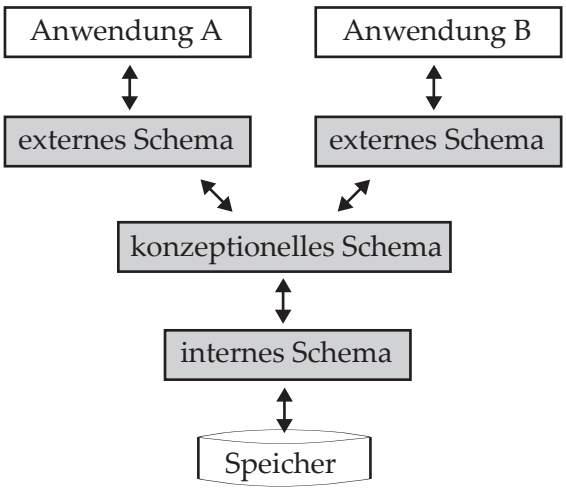

Abbildung 2.13: Drei-Schichten-Architektur von Datenbanksystemen, nach [Leser / Naumann, 2007, 85]

und wie diese miteinander in Beziehung stehen. Die Trennung von der internen Sicht gewährleistet eine *physische Datenunabhängigkeit*. Das konzeptionelle Schema kann so unabhängig von technischen Details entwickelt werden. Umgekehrt hat die Änderung von technischen Details, bspw. der Austausch einer Festplatte oder die Änderung von Speicherorten keinen Einfluss auf die logische Repräsentation der Daten.

Die *externe Sicht* mit dem *Exportschema* modelliert ebenfalls Daten. Allerdings wird nicht die komplette Anwendungsdomäne, also alle Daten im konzeptionellen Schema, breücksichtigt, sondern ein Ausschnitt aus dieser, der speziell auf die zugreifende Anwendung zugeschnitten ist. Außerdem können Zugriffsbeschränkungen vorgenommen werden. Die Trennung zwischen externer und konzeptioneller Sicht erlaubt eine *logische Datenunabhängigkeit*. Das konzeptionelle Schema kann unabhängig vom Exportschema angepasst und verändert werden und ungekehrt.

Für die Umsetzung der Integration über die Datenhaltungsschicht werden im folgenden zwei Szenarien vorgestellt: zum einen die Verwendung einer *materiell integrierten Datenbank* und zum anderen die Verwendung von *verteilten Datenbanken*.

2.6.3.2 Integration über monolithische Datenbank

Bei dieser Integrationsmethode benutzen die Teilsysteme gemeinsam eine einzige materielle Datenbank, deren konzeptionelles Schema folglich alle relevanten

Datenrelationen enthalten muss. Eine solche Datenbank kann aus den bestehenden Datenbanken durch Werkzeuge zur Extraktion, Transformation und zum Laden (→ETL-Tools) in die neue Datenbank befüllt werden [Speyerer, 2005, 14]. Die Erstellung eines konzeptionellen Schemas, dass alle benötigten Datenkonzepte bereitstellt, kann sich jedoch sehr aufwändig gestalten. Die Abbildung2.13 verdeutlicht diesen Ansatz grafisch.

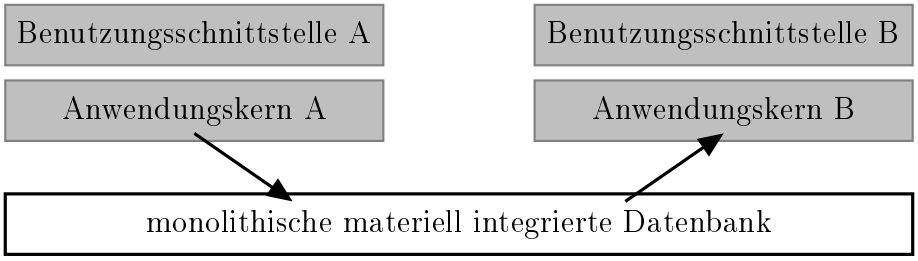

Abbildung 2.14: Integration über eine materiell integrierte Datenbank. Erstellt nach [Keller, 2002, 67].

2.6.3.3 Integration über verteilte Datenbanken

Beim Einsatz von verteilten Datenbanken liegen die Daten nicht in einem einzigen zentralen System vor. Vielmehr sind sie physisch und logisch auf verschiedenen Systemen verteilt. Die Gründe für eine solche Verteilung sind u.a: [Leser / Naumann, 2007, 54]

- Verteilung der Anfragelast auf mehrere Systeme und damit Performanceverbesserungen
- Ausfallsicherheit und höhste Verfügbarkeit durch redundante Speicherung
- Schutz vor Datenverlust durch technische Unfälle

In Abschnitt 2.4.1, S. 17 wurde dargelegt, welche Probleme eine ungewollte Verteilung für die Integration mit sich bringt. Im Fall der verteilten Datenbanken ist die Verteilung jedoch durch das Design gewollt und geplant und wird daher strikt kontrolliert. Die einzelnen Datenquellen geben ihre Autonomie (s. Abschnitt 2.4.2, S. 19) vollständig auf. Ihre Schemata müssen sich mit dem globalen Schema der verteilten Datenbank vollständig zusammenführen lassen. Dadurch können schwerwiegende Probleme, wie strukturelle oder semantische Heterogenität vermieden werden. [Leser / Naumann, 2007, 91ff]

Abbildung 2.15 zeigt die Vier-Schichten-Architektur von verteilten Datenbanksystemen. Oberhalb der lokalen konzeptionellen Schemata befindet sich das

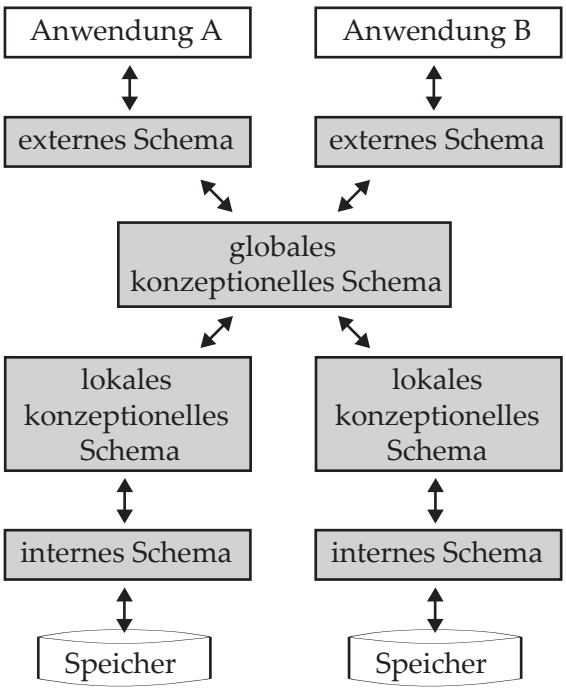

Abbildung 2.15: Vier-Schichten-Architektur für verteilte DBMS, nach [Leser / Naumann, 2007, 92]

globale konzeptionelle Schema. Es modelliert aus den lokalen konzeptionellen Schemata ein vollständiges Datenschema für die gesamte Anwendungsdomäne. Die lokalen konzeptionellen Schemata dürfen daher keine Datenmodellheterogenität zum globalen konzeptionellen Schema aufweisen. In der Praxis bedeutet das oft, dass zuerst das globale konzeptionelle Schema entworfen wird, und anschließend im sog. *Verteilungsentwurf* festgelegt wird, welche Daten wo gespeichert werden, und wie die entsprechenden lokalen konzeptionellen Schemata modelliert werden. Die externen Schemata für die Anbindung an Anwendungen beziehen sich bei verteilten Datenbanken auf das globale konzeptionelle Schema und nicht auf einzelne lokale Schemata. [Leser / Naumann, 2007, 92] Die Verteilung der Daten auf die einzelnen lokalen Datenquellen ist für die externen Schemata also transparent (*Verteilungstransparenz*). Für die Anwendung und den Nutzer soll der Eindruck entstehen, dass sie mit einer einzigen monolithischen Datenbank arbeiten [Leser / Naumann, 2007, 91]. Die Verwendung von verteilten Datenbanken ist also eine Methode der virtuellen Integration.

Für die Integration bereits existierender Datenquellen ist dieser Ansatz daher aufgrund der zu erwartenden Heterogenitätsprobleme ungeeignet. Bereits existierende Datenbestände benötigen eine Integrationstechnik, die den einzelnen

Datenquellen deutlich mehr Autonomie in ihren Datenmodellen und Datenrepräsentationen einräumt. Ein oft verwendeter Ansatz dafür ist der Einsatz von *föderierten Datenbanksystemen* (FDBMS). Diese bauen auf dem Prinzip der verteilten Datenbanken auf. In Abschnitt 2.7.3, S. 49 wird detaillierter auf FDBMS eingegangen.

2.7 Überblick über verbreitete Integrationstechniken

Die folgenden Abschnitte bieten zum Abschluss des Grundlagen-Kapitel einen Überblick über weit verbreitete Integrationstechniken. Sie entstammen unterschiedlichen Aufgaben- und Anwendungsbereichen und stellen einen Querschnitt über in der Praxis bewährte Integrationsmethoden dar.

2.7.1 Web Services

Die Integration über die Verwendung von Web Services gewinnt immer mehr an Bedeutung und soll deshalb etwas ausführlicher behandelt werden. Web Services basieren auf XML. Daher wird zunächst XML kurz beleuchtet, bevor auf die Merkmale und Funktionsweise der Web Services näher eingegangen wird.

2.7.1.1 XML als Basisformat

Die *Extensible Markup Language* (XML) ist eine vom → World Wide Web Consortium (W3C) standardisierte Beschreibungssprache, mit der sich beliebige Datenstrukturen abbilden und austauschen lassen. Wie auch die Hypertext Markup Language (HTML) ist XML aus der Standard Generalized Markup Language (SGML) abgeleitet und verwendet in Form einer Baumstruktur verschachtelte Tags, um die Datenstrukturen abzubilden [vgl. Burbiel, 2007, 123]. Abweichend von HTML ist der Grundgedanke von XML die Trennung von Inhalt und Darstellung eines Dokuments. Ein XML-Dokument sagt nichts darüber aus, wie die enthaltenen Daten dargestellt werden sollen [vgl. Speyerer, 2005, 24]. XML kann sowohl in Form von Dateien vorliegen, als auch aus anderen Quellen stammen, bspw. aus einer Datenbank oder als Antwort eines

→RPC. Da praktisch jede Software XML einlesen und interpretieren und verarbeiten kann, ist es auch das verbreiteste Format für Schnittstellen, nicht nur im Bereich der Web Services. [vgl. Burbiel, 2007, 142]

```
<?xml version="1.0" standalone="yes"?>
<dataSet>
  <tblPersonen>
    <Id>123</Id>
    <Vorname>Peter</Vorname>
    <Nachname>Müller</Nachname>
    <Geburtstag>1967-07-08</Geburtstag>
  </tblPersonen>
  <tblPersonen>
    <Id>456</Id>
    <Vorname>Max</Vorname>
    <Nachname>Meier</Nachname>
    <Geburtstag>1944-07-12</Geburtstag>
  </tblPersonen>
</dataSet>
```

Tabelle 2.3: Beispiel für ein wohlgeformtes XML-Dokument

XML enthält Mechanismen um zu prüfen, ob das vorliegende XML *wohlgeformt* und *gültig* ist. Ein XML-Dokument ist wohlgeformt, wenn alle formalen Regeln von XML eingehalten sind. Bspw. muss ein Wurzelelement vorhanden sein, dass alle anderen Elemente beinhaltet; es muss zu jedem geöffneten Tag ein schließendes Tag existieren, und die Groß- und Kleinschreibung beachtet werden (XML ist case-sensitive)[1]. Dies lässt sich mit geeigneten Programmen, sog. →Parsern, herausfinden. Diese prüfen das Dokument bereits vor dem Einlesen auf formale Gültigkeit, sodass bereits vor der weiteren Verarbeitung der Daten im Dokument Aussagen über dessen Wohlgeformtheit getroffen werden können [Speyerer, 2005, 24]. Durch den Einsatz von *Document Type Definitions* (DTD), bzw. XML-Schemata können XML-Dokumente auch validiert werden, d.h. es kann geprüft werden, ob die Strukturen der enthaltenen *Daten* den semantischen und syntaktischen Vorgaben entsprechen [vgl. Speyerer, 2005, 24]. Eine DTD informiert das verarbeitende System darüber,

> „welche Elementtypen die XML-Datei beinhaltet, welche Werte diese Elementtypen annehmen dürfen, welche Attribute erlaubt sind und welche Werte diese Attribute annehmen dürfen.“ [Burbiel, 2007, 125]

Im Mai 2001 wurde die DTD durch die Spezifikation des *XML-Schema* des →W3C abgelöst, die aktuelle Spezifikation ist XML-Schema vom Oktober

[1] Für eine vollständige Liste aller Kriterien eines wohlgeformten XML-Dokuments sei auf die entsprechende Recommandation des W3C verwiesen, erreichbar unter: http://www.w3.org/TR/2006/REC-xml-20060816/ (letzter Aufruf: 2008-08-27).

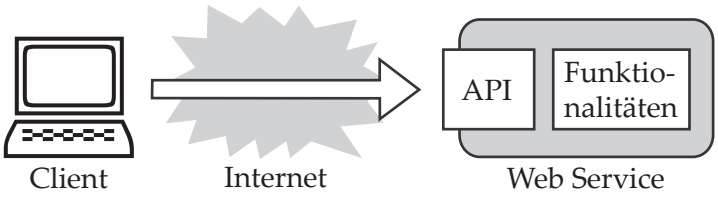

Abbildung 2.16: Verbildlichung der Definition von Web Service durch [Speyerer, 2005, 25]

2004 W3C [2004b]. XML-Schemata sind selbst wiederum XML-Dokumente, die jedoch nur vordefinierte Elemente enthalten dürfen [Burbiel, 2007, 128]. Im Gegensatz zu den DTD sind sie flexibler einsetzbar. Sie können z.B. auch komplexe Datentypen abbilden, die aus mehreren Elementen komplexer oder einfacher Datentypen bestehen [Burbiel, 2007, 129]. XML-Schemata liegen immer in eigenständigen Dateien vor. Sie können sowohl selbst erstellt werden, es existiert jedoch auch eine große Anzahl von ihnen frei verfügbar im Internet [Speyerer, 2005, 24]. Für die meisten Anwendungsfälle ist bspw. ausreichend, die Schemata des W3C zu verwenden[2].

2.7.1.2 Definition und Abgrenzung von Web Services

Web Services sind eine Technik der Integration über die Anwendungsschicht, d.h. es werden grundlegend Funktionen auf einem entfernten System aufgerufen und deren Rückgaben in das aufrufende System zurück übertragen. [Speyerer, 2005] stellt dar, wie stark die konkreten Definitionsansätze zum Begriff → Web Services (WS) in der Literatur differieren. In seinem eigenen Ansatz werden WS definiert als

> „ ... in sich geschlossene, selbstbeschreibende, modulare Web-Module, welche eine Netzwerkschnittstelle enthalten, über die sie veröffentlicht, lokalisiert und aufgerufen werden können."
> [Speyerer, 2005, 25]

Von dieser Definition ausgehend leitet [Speyerer, 2005, 25f] die folgenden Eigenschaften von WS ab:

Kapselung und *Modularität*: Die Funktionalitäten werden in separaten und wiederverwendbaren Softwaremodulen gekapselt. Sie erlauben so die Nutzung

[2] Erreichbar unter http://www.w3.org/1999/XMLSchema - dieses Schema wird auch in den folgenden Beispielen verwendet. Letzter Aufruf am 11.05.2009

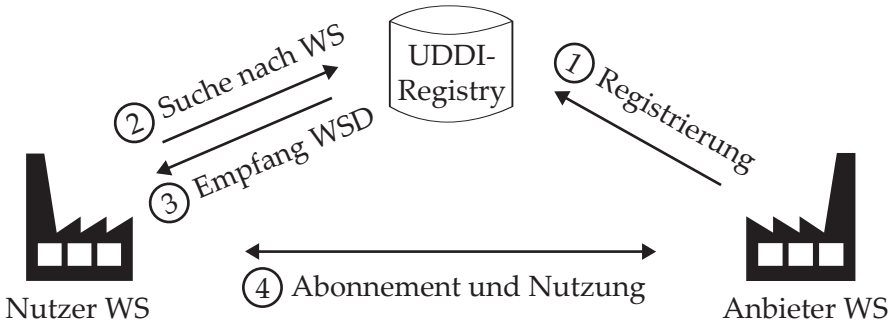

Abbildung 2.17: zeigt den Ablauf der Veröffentlichung, Abonnierung und Nutzung von WS bei der Verwendung eines UDDI-Verzeichnisses, in Anlehnung an [Speyerer, 2005, 27].

von WS ohne Preisgabe von Implementierungsdetails (*Information-Hiding-Prinzip*) und eine flexible Rekombination der Module zu neuen Funktionalitäten.

Selbstbeschreibung: Mit Hilfe der *Web Services Description Language* (WSDL) können erwartete Ein- und Ausgabeparameter, sowie zu verwendende Datenformate den Nutzern von WS in standardisierter und maschinenlesbarer Form bekannt gegeben werden.

Verfügbarkeit: Die Anbieter von WS veröffentlichen ihre Dienste in einem UDDI-Verzeichnis (Service Registry). UDDI steht für *Universal Description Discovery and Integration* und ist mit einem Branchenbuch vergleichbar. Es führt WS mit ihren relevanten WSDL-Spezifikationen (Schnittstellen, Datenformate, URI) und meist weiteren Angaben zum Anbieter und zur Branche. Ein solches Verzeichnis kann allgemein zugänglich sein, oder auch nur einem beschränkten Nutzerkreis zur Verfügung stehen. Die Nutzer können dann in diesem Verzeichnis nach für sie geeigneten WS suchen und diese „abonnieren“.

Auffindbarkeit: Der Nutzer eines WS muss nicht wissen, wo sich der Dienst physikalisch befindet. Die benötigten Informationen stehen ihm über UDDI-Verzeichnisse zur Verfügung. Dies steht in engem Zusammenhang mit der Eigenschaft der *Verfügbarkeit*. Da ein WS „registriert“ wird, ist es sogar möglich, vorher gänzlich unbekannte Anbieter von Funktionalitäten, und damit potentielle Geschäftspartner, automatisiert zu finden und mit ihnen zu interagieren. Natürlich sind WS bei Kenntnis der Sepzifikationen auch direkt ohne die Suche in einem Verzeichnis auffindbar, bspw. wenn ein WS nur unternehmensintern genutzt werden soll.

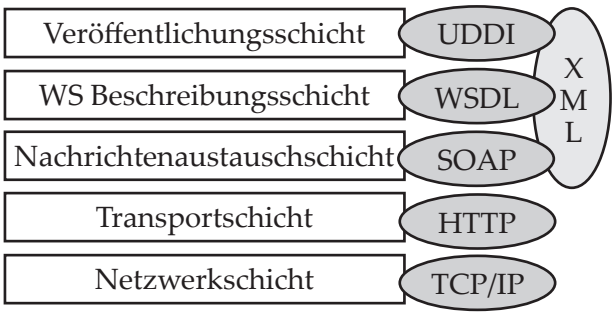

Abbildung 2.18: Web Service Stack (vereinfacht), nach [W3C, 2004a].

Standardisierung: Durch den Einsatz von standardisierten Technologien, wie XML, HTTP oder TCP/IP (siehe Abschnitt 2.7.1.3, S. 44) und die Standardisierungsbemühungen bei den übrigen Technologien (SOAP, WSDL, UDDI) wird eine weitgehende Unabhängigkeit von Plattformen und Programmiersprachen erreicht.

2.7.1.3 Funktionsweise von Web Services

In Anlehnung an das → OSI Schichtenmodell werden die für WS verwendeten Standard- bzw. im Standardisierungsprozess befindlichen Technologien in einem Schichtenmodell, dem *Web Service Stack*, angeordnet (siehe Abbildung 2.7.1.3).

Die erste Schicht ist die *Netzwerkschicht*. Sie fasst die Bitübertragungs-, die Sicherungs- und die Vermittlungsschicht des OSI-Modells zusammen und dient der Adressierung und Weiterleitung von Paketen zwischen beliebigen Endpunkten in Netzwerken mittels des → Internet Protocol (IP) und des → Transmission Control Protocol (TCP) [Speyerer, 2005, 27].

Darauf aufbauend schließt sich die *Transportschicht* an. Sie ist für den Datenaustausch zwischen Programmen verantwortlich. Zum Einsatz kommt fast immer das → Hypertext Transfer Protocol (HTTP), da es auch von restriktiveren Firewalls in Unternehmen meist zugelassen wird [vgl. Strobel, 2006, 27]. Es sind jedoch auch eine Vielzahl anderer Transportprotokolle einsetzbar (FTP, SMTP, proprietäre Protokolle, etc.) [W3C, 2004a].

In der *Nachrichtenaustausch-Schicht* werden die eigentlichen Daten unter Verwendung des *Simple Object Access Protocol* (SOAP) erzeugt und ausgetauscht. SOAP basiert auf XML und beschreibt ein *Nachrichtenformat zur Übermittlung von Daten, die üblicherweise bei einer Operation anfallen* (zum Beispiel Klassen und Methodenname, Parameter, Rückgabewerte) [Speyerer, 2005, 28].

Die Nachrichten bestehen jeweils aus dem *SOAP Envelope*, einer Art „Umschlag“, der mitteilt, dass es sich um eine SOAP-Nachricht handelt, welches XML-Schema (siehe Abschnitt 2.7.1.1, S. 40) verwendet wird, und in weiteren Parametern bspw. Angaben zur verwendeten Zeichencodierung macht. Außerdem enthält der Envelope einen optionalen *SOAP-Header* und einen obligatorischen *SOAP-Body*. Der Header enthält verschiedene Steuerinformationen und im Body steht die zu versendende Nachricht. Beide Elemente können beliebig viele, dem Namensraum entsprechende, gültige XML-Unterknoten enthalten. [W3C, 2003a]

Mit SOAP-Nachrichten können zwei verschiedende Austauschszenarien umgesetzt werden. Zum einen können Nachrichten ausgetauscht werden, die keine besonderen Rückgaben an den Sender der Nachricht bedingen, bspw. Bestellungen oder Bestätigungen. Man spricht dann vom *Document-Style*. Zum anderen können SOAP-Nachrichten im *RPC-Style* verwendet werden, also zum Aufruf einer entfernten Funktionalität. Dann wird zuerst ein *SOAP-Request* mit den benötigten Argumenten und der aufzurufenden Funktionalität gesendet, und nach der Abarbeitung der Funktionalität eine *SOAP-Response* mit den angefragten Daten an den aufrufenden Sender zurück übermittelt. [vgl. Speyerer, 2005, 29]

In einer SOAP-Nachricht können die Datentypen der enthaltenen Daten durch eine *Encoding-Angabe* festgehalten und so aus der unterschiedlichen Darstellung von Datentypen in verschiedenen Plattformen und Programmiersprachen resultierende Probleme umgangen werden [Speyerer, 2005, 28] – siehe auch Tabelle 2.4.

Auf der *WebService-Beschreibungsschicht* werden WS unter Verwendung der *Web Services Description Language* (WSDL) beschrieben. Die WSDL ist vom → W3C standardisiert und beschreibt WS als eine Menge von Endpunkten mit jeweils einer Menge von Operationen und erwarteten Ein- und Ausgabeparametern, die den Zugang zu der jeweiligen Funktionalität beschreiben. Zur Beschreibung wird wiederum ein XML-Format verwendet. W3C [2001]. Die WSDL abstrahiert dabei von den zugrunde liegenden Kommunikationsprotokollen. Die Spezifikationen der Funktionalität werden also unabhängig von Plattformen oder Protokollen definiert und sind daher überall flexibel einsetzbar. Die definierten abstrakten Endpunkte werden durch ein *Binding*, d.h. die Zuweisung einer konkreten Netzwerkadresse und eines Kommunikationsprotokolls, zu sog. *Ports*, also adressierbaren Endpunkten im Netz. Eine Sammlung von solchen Ports ergibt dann einen WS. [vgl. Speyerer, 2005, 30] In Tabelle 2.5 wird eine beispielhafte WSDL-Spezifikation angeführt, die Funktionen der einzelnen Zeilen können der Tabellenunterschrift entnommen werden.

```
1
2 <soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
3       soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding"
4       xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
5       xmlns:xsd="http://www.w3.org/1999/XMLSchema">
6   <soap:Header>
7     <t:Transaction xmlns:t="http://example.com/transaction"
8           soap:mustUnderstand="true">
9       <transactionID>1234</transactionID>
10      <transactionTimeStamp>1127383672</transactionTimeStamp>
11    </t:Transaction>
12  </soap:Header>
13  <soap:Body>
14    <m:GetAvailability xmlns:m="http://example.com/ws/getAvailability">
15      <productName xsi:type="xsd:string">Product XYZ</productName>
16      <quantity xsi:type="xsd:int">50</quantity>
17      <deliveryDate xsi:type="xsd:dateTime">2008-08-14</deliveryDate>
18      <companyID xsi:type="xsd:int">1234</companyID>
19    </m:GetAvailability>
20  </soap:Body>
21 </soap:Envelope>
```

Tabelle 2.4: SOAP-Request mit HTTP im RPC-Style. Aus [Speyerer, 2005, 30]

Auf der obersten Schicht, der *Veröffentlichungsschicht*, können die WS in UDDI-Verzeichnissen veröffentlicht und anderen Nutzern zugänglich gemacht werden. UDDI steht für *Universal Description, Discovery and Integration* und wurde von Microsoft, IBM und Ariba entwickelt, um Geschäftspartner und deren WS schnell und einfach zu finden und elektronische Geschäftsbeziehungen über die Nutzung von WS aufzubauen [Speyerer, 2005, 31]. Das Verzeichnis kann sowohl zur öffentlichen Publikation eines WS für alle Verzeichnisnutzer, als auch zur unternehmensinternen Veröffentlichung genutzt werden OASIS [2002]. Das UDDI-Projekt wird betreut von der Organization for the Advancement of Structured Information Standards (OASIS), einem globalen Konsortium zur Entwicklung und Durchsetzung von standardisierten Technologien für das E-Business [Speyerer, 2005, 31]. Zu weiteren Details und der logischen Repräsentation der Informationen in UDDI-Verzeichnissen sei an dieser Stelle auf die Ausführungen in [Speyerer, 2005, 31f] verwiesen.

2.7.1.4 Leistungsfähigkeit

Das stärkste Argument für den Einsatz von WS für Integrationslösungen ist die flexible, lose Kopplung der Integrationspartner. Die Kommunikation und Kooperation kann schnell eingeleitet und flexibel gehandhabt werden. Vor allem für die externe horizontale Integration zwischen verschiedenen Unternehmen sind WS optimal geeignet. Geschäftspartner können effizient und nach ge-

```
1  <?xml version="1.0"?>
2  <definitions name="StockQuote"
3      targetNamespace="http://example.com/stockquote.wsdl"
4      xmlns:tns="http://example.com/stockquote.wsdl"
5      xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
6      xmlns:xsd1="http://example.com/stockquote.xsd"
7      xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
8      xmlns="http://schemas.xmlsoap.org/wsdl/">
9
10   <message name="GetTradePriceInput">
11     <part name="tickerSymbol" element="xsd:string"/>
12     <part name="time" element="xsd:timeInstant"/>
13   </message>
14
15   <message name="GetTradePriceOutput">
16     <part name="result" type="xsd:float"/>
17   </message>
18
19   <portType name="StockQuotePortType">
20     <operation name="GetTradePrice">
21       <input message="tns:GetTradePriceInput"/>
22       <output message="tns:GetTradePriceOutput"/>
23     </operation>
24   </portType>
25
26   <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
27     <soap:binding style="rpc"
28         transport="http://schemas.xmlsoap.org/soap/http"/>
29     <operation name="GetTradePrice">
30       <soap:operation soapAction="http://example.com/GetTradePrice"/>
31       <input>
32         <soap:body use="encoded" namespace="http://example.com/stockquote"
33             encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
34       </input>
35       <output>
36         <soap:body use="encoded" namespace="http://example.com/stockquote"
37             encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
38       </output>
39     </operation>
40   </binding>
41
42   <service name="StockQuoteService">
43     <documentation>Stock Quote Web Service Example WSDL</documentation>
44     <port name="StockQuotePort" binding="tns:StockQuoteBinding">
45       <soap:address location="http://example.com/stockquote"/>
46     </port>
47   </service>
48 </definitions>
```

Tabelle 2.5: Beispiel einer WSDL-Spezifikation eines Web Service, der Angaben zu Aktienkursen liefert. In den Zeilen 26-40 wird das Binding für einen Port „GetTradePrice“ beschrieben, der unter der URL „http://example.com/GetTradePrice“ erreichbar wäre. In den Zeilen 19-24 werden die erwarteten Anfrage- und Antwortnachrichten definiert, deren Argumente wiederum in den Zeilen 10-17 spezifiziert werden. Aus [W3C, 2001]

nau definierten Kriterien automatisch gewählt werden, die Supply Chain wird nachhaltig optimiert, der Kunden- und Geschäftspartnerstamm wird breiter und stabiler. Zudem überwindet die Plattformunabhängigkeit von WS viele der bisherigen technischen Barrieren. Ein weiterer großer Vorteil liegt in der Modularisierung. Sie erlaubt es, genau umrissene Funktionalitäten abzugrenzen, sie unabhängig zu pflegen und zu verbessern und sie gezielt den Nutzern verfügbar zu machen. Die sich dadurch ausprägende Spezialisierung erlaubt es beiden Seiten, auf qualitativ hochwertige Funktionen und Ergebnisse zu bauen, anstatt selbst Funktionalitäten zu implementieren, deren Güte aufgrund mangelnder Fachkenntnisse zu wünschen übrig lässt, oder deren Erstellung einen unangemessen hohen Aufwand mit sich bringt.

Eines der Haupthemmnisse beim Einsatz von WS sind derzeit noch *Sicherheitsbedenken*. Vor allem werden fehlende Standardisierungen in sicherheitsrelevanten Bereichen (Verschlüsselung, Signaturen, Authentizität, Vertraulichkeit, Verbindlichkeit, usw.) bemängelt. Hierzu gibt es von Seiten des W3C und der OASIS vielfache Standardisierungsbemühungen, die aber als für den produktiven Einsatz größtenteils noch nicht nutzbar angesehen werden. [Speyerer, 2005, 37] Ein weiteres Problem ist die Koordinierung und Kontrolle von Geschäftsprozessen über mehrere WS hinweg, z.B. für Transaktionsmanagement u.ä. Hierzu gibt es bereits akademische Ansätze, die aber ebenfalls noch nicht für den Einsatz in Produktivsystemen bereit sind [Speyerer, 2005, 38].

2.7.2 Portale

Die Portaltechnik kommt vor allem für Integrationssysteme zum Einsatz, die über das Internet weltweit, oder das → Intranet firmenintern erreichbar, und mittels eines Browsers bedienbar sind. Möglich ist bspw. die Einbettung von Benutzeroberflächen verschiedener Anwendungen in Frames von Webseiten, wie in Abbildung 2.19 dargestellt. Die Integration kann aber auch über das Einbinden von HTML-Fragmenten oder die Verwendung von JavaScript umgesetzt werden.

2.7.2.1 Leistungsfähigkeit

Portale bieten eine einfache Möglichkeit, verschiedene Anwendungen über die Benutzeroberfläche zu integrieren (siehe Abschnitt 2.6.1, S. 32). Die Vorteile der Portaltechnik sind:

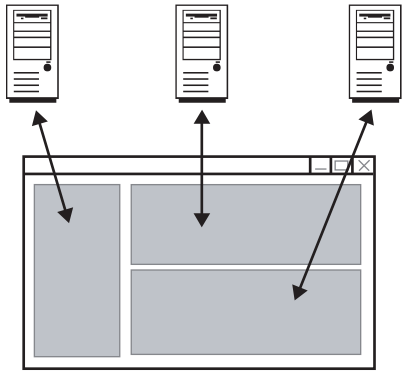

Abbildung 2.19: Portalintegration über Frames, nach [Keller, 2002, 62]

- Das Portal bietet einen *zentralen Zugang* zu allen angeschlossenen Anwendungen.
- Informationen aus verschiedenen Anwendungen können in *neuen Zusammenhängen* betrachtet und neue Informationen gewonnen werden.
- Inhalte können *personalisiert* werden, d.h. unterschiedliche Benutzer bekommen unterschiedliche relevante Informationen.
- Im Portal können *übergreifende Operationen* schnell und übersichtlich getätigt werden.
- Das Portal ist leicht und *plattformunabhängig* zugänglich, und über einen Browser *komfortabel* zu bedienen.
- Der Erstellungsaufwand ist relativ gering, da bereits fertige Anwendungskomponenten wiederverwendet werden.

2.7.3 Föderierte Datenbanksysteme

Föderierte Datenbanksysteme (FDBMS) speichern, wie verteilte Datenbanken (Abschnitt 2.6.3.3, S. 38) auch, Daten auf physisch und logisch verteilten Systemen. Für die Definition der externen Schemata zum Zugriff auf das FDBMS wird auch hier ein *globales konzeptionelles Schema* verwendet. Im Unterschied zu verteilten Datenbanken entsteht dieses aber mit der Absicht „eine integrierte Sicht auf existierende und heterogene Datenbestände zu bieten“ [Leser / Naumann, 2007, 94]. Die einzelnen Datenquellen behalten ein hohes Maß an Autonomie, damit sie die bereits vorhandenen Daten in gewohnter Weise weiter verwalten können[3]. Die Verteilung und Heterogenität der Datenquellen in

[3] Daher stammt auch der Name „Datenbankföderation“ - Eine Föderation bezeichnet einen losen Verbund mehrerer unabhängiger Teile zur Zusammenarbeit.

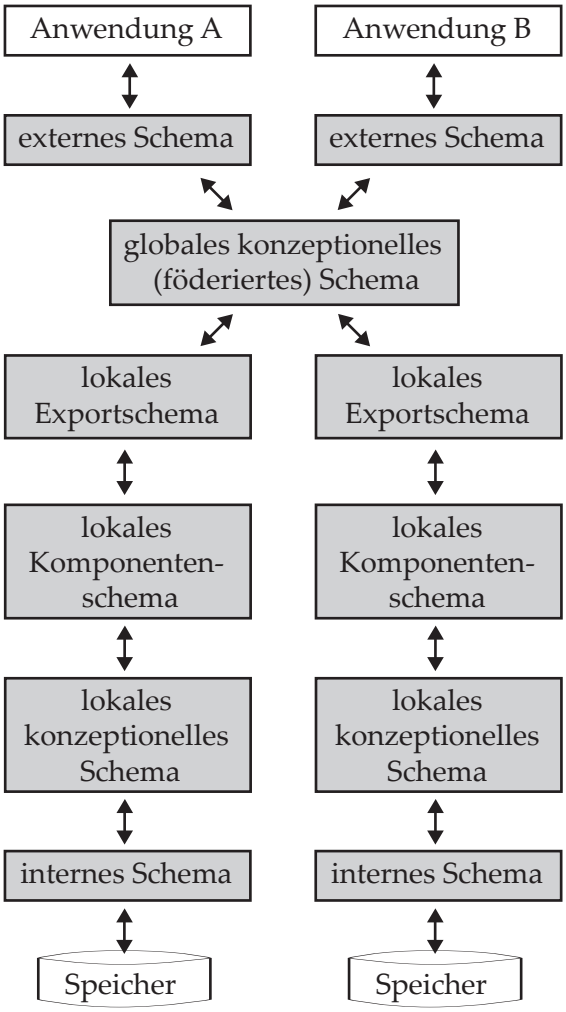

Abbildung 2.20: Fünf-Schichten-Architektur von FDBMS, nach [Leser / Naumann, 2007, 95]

einer Föderation ist also nicht gewollt, sondern muss als gegeben hingenommen werden. [Leser / Naumann, 2007, 94]

Die Architektur einer Datenbankföderation gliedert sich in fünf Schichten, von denen jede durch eigene Schemata charakterisiert wird (s. Abbildung 2.20). Von der Vier-Schichten-Architektur der verteilten Datenbanken, die in Abschnitt 2.6.3.3, S. 38 erläutert wird, unterscheidet sich die Architektur der FDBMS hauptsächlich durch die Existenz des *lokalen Komponentenschemas*. Dieses Schema modelliert die Inhalte des lokalen konzeptionellen Schemas in das globale konzeptionelle Schema. Es entkoppelt also das *lokale* konzeptionelle Schema vom *globalen* konzeptionellen Schema, überbrückt somit die Heterogenitäten zwischen den Beiden und stellt die weitgehende *Designautonomie* der lokalen Datenquellen sicher.

Weiterhin weisen die lokalen Datenquellen ein eigenes *lokales Exportschema* auf. Das Exportschema regelt, welche Teilmenge des lokalen Komponentenschemas nach außen für wen sichtbar ist. Die lokale Datenquelle weist also auch *Zugriffsautonomie* auf und ermöglicht einzelnen Anwendungen, auch ohne die Nutzung der vom globalen konzeptionellen Schema abgeleiteten externen Schemata, den Zugriff auf lokale Daten.

Das globale konzeptionelle Schema wird in der Literatur auch als *föderiertes Schema* oder integriertes Schema bezeichnet. Die von ihm abgeleiteten externen Schemata bieten, je nach deren Ausgestaltung, den eigentlichen integrierten Zugang auf alle föderierten Datenquellen. Das Datenmodell, welches das föderierte Schema nutzt, muss auch von allen lokalen Datenquellen in den Exportschemata verwendet werden. Es wird daher als das *kanonische Datenmodell*. Wenn lokale Datenquellen in ihren konzeptionellen Schemata andere Datenmodelle verwenden, so müssen lokale Komponentenschemata diese Datenmodelle in das kanonische Datenmodell überführen. [Leser / Naumann, 2007, 94ff]

Das föderierte Schema entsteht in der Praxis erst dann, wenn die lokalen Exportschemata bereits vorliegen. Zur Erzeugung des föderierten Schemas gibt es zwei Ansätze: Entweder es wird aus den einzelnen Exportschemata abgeleitet und zusammengestellt – man spricht dann von *Schemaintegration* – oder aber es wird unabhängig von den lokalen Exportschemata entworfen und stellt eine Sammlung von Transformationsvorschriften für die Exportschemata dar. Diese Methode wird als *Schema Mapping* bezeichnet. Beide Verfahren werden im Folgenden kurz vorgestellt.

2.7.3.1 Schemaintegration

Der Vorgang der Schemaintegration bezeichnet die Erstellung des zentralen, für die Integration zuständigen *föderierten Schemas* durch Ableitung aus den vorhandenen Exportschemata der einzelnen Datenquellen. Voraussetzung für die Ableitung des föderierten Schemas ist es, dass alle lokalen Datenquellen in ihren Exportschemata das kanonische Datenmodell nutzen. Trotz Verwendung des gleichen Datenmodells liegt zwischen den lokalen Exportschemata aller Vorraussicht nach strukturelle und semantische Heterogenität vor. Die Datenobjekte überlappen sich teilweise, bilden Repräsentationen desselben Objekts verschieden ab, kommen mehrfach vor, bilden verschiedene Teilbereiche des Objekts ab usw.

Die Schemaintegration möchte die zu repräsentierenden Datenobjekte möglichst umfassend und konsistent integrativ verfügbar machen und definiert dazu mehrere Anforderungen [Leser / Naumann, 2007, 117f]:

- *Vollständigkeit*: Das föderierte Schema soll die vollständige Vereinigung aller Domänen der lokalen Datenquellen sein, d.h. alle Konzepte, sowie deren Informationen und Beziehungen untereinander sollen vorhanden sein.
- *Korrektheit*: Jedes Konzept des föderierten Schemas muss mindestens zu einem Konzept in einem lokalen Schema bedeutungsgleich sein. Außerdem dürfen keine Beziehungen von Konzepten und Elementen des föderierten Schemas im Widerspruch zu denen in den lokalen Schemata stehen.
- *Minimalität*: Semantisch gleiche Konzepte sollen nur einmal, aber aus der Sicht aller lokalen Datenquellen vollständig und korrekt, repräsentiert sein.
- *Verständlichkeit*: Bedeutungen und Namen von Relationen und Konzepten müssen für Entwickler und Benutzer verständlich sein und sollten den ursprünglichen Bezeichnungen in den lokalen Schemata möglichst ähnlich gehalten werden.

Diese Anforderungen, insbesondere die der Verständlichkeit, zeigen dass eine vollautomatische Schemaintegration zumindest derzeit noch nicht möglich ist. In [Leser / Naumann, 2007] wird dazu bemerkt:

> „Schemaintegration bleibt eines der schwierigsten Probleme der Informationsintegration. Trotz mindestens 30 Jahren Forschung gibt es bis heute keinen Vorschlag, der genügend Flexibilität und Ausdrucksstärke besitzt, um mit realen (d.h. großen, unübersichtlichen und auf allen Ebenen hochgradig heterogenen) Schemata zufriedenstellend umzugehen. Schemaintegration sollte daher eher als Kunst denn als systematische Tätigkeit begriffen werden.“ [Leser / Naumann, 2007, 122]

Aufgrund der komplexen und schwer zu verallgemeinernden Bedingungen soll im Rahmen dieser Arbeit lediglich ein grobes Vorgehensmodell für die Schemaintegration angeführt werden. Für Einzelheiten zu speziellen Verfahren sei auf entsprechende Fachpublikationen wie [Conrad, 1997] oder [Batini u. a., 1986] verwiesen.

Der Prozess der Schemaintegration wird nach [Batini u. a., 1986] in die vier Arbeitsschritte Vorintegration, Schemavergleich, Schemaangleichung und Sche-

mafusion eingeteilt. Bei der *Vorintegration* werden zunächst die zu integrierenden lokalen Datenquellen ausgewählt und weitere Informationen, wie Dokumentationen u.ä. zu diesen ausgewertet, um ein möglichst umfassendes Verständnis der Datenquellen und ihrer Schemata zu erlangen. Anschließend werden die Datenquellen nach ihrer Relevanz in eine Reihenfolge gebracht. In dieser Reihenfolge werden sie später verglichen und zusammengefasst. Durch die Festlegung der Reihenfolge können wichtigere Teilschemata einen größeren Einfluss auf das Gesamtergebnis bewirken.

Beim anschließenden *Schemavergleich* werden *Korrespondenzen* zwischen den Schemata ermittelt, also Aussagen darüber getroffen, welche Ausdrücke miteinander äquivalent, ineinander enthalten, einander überlappend oder unverbunden sind. Korrenspondenzen gibt es auf verschiedenen Ebenen des Datenmodells: von einfachen Attributkorrespondenzen, über Schemakorrespondenzen bis hin zu Anfragekorrespondenzen. Für die Auswertung von einfachen Attributkorrespondenzen existieren verschiedene automatische Verfahren, die inter dem Begriff *Schema Matching* zusammengefasst werden. Diese hier darzustellen würde den Rahmen dieser Arbeit sprengen. Es sei daher auf die Ausführungen in [Leser / Naumann, 2007, 143-157] zum Schema Matching verwiesen. Weiterhin werden beim Schemavergleich eventuelle Konflikte in den Schemata aufgedeckt, wie bspw. synonyme Bezeichner für unterschiedliche Eigenschaften oder inkompatible Strukturierungen der Datenobjekte.

Im Zuge der *Schemaangleichung* werden die gefundenen Probleme und Konflikte nun ausgeglichen, etwa durch Umbenennung von Eigenschaften oder durch Umstrukturierungsmaßnahmen. Für jedes Ausgangsschema entsteht somit ein transformiertes Schema. In den transformierten Schemata werden dann identische Konzepte auch identisch abgebildet.

Abschließend werden in der *Schemafusion* die transformierten Schemata zu einem einheitlichen Schema fusioniert. Um insbesondere die Anforderung der Verständlichkeit zu erfüllen, können weitere Umstrukturierungen am integrierten Schema erforderlich werden [Leser / Naumann, 2007, 119].

2.7.3.2 Schema Mapping

Im Unterschied zur Schemaintegration, bei der ein komplett neues Schema entwickelt wird, bezeichnet Schema Mapping eine Menge von *Vorschriften zur Transformation von Daten zwischen existierenden Schemata* [vgl. Leser / Naumann, 2007, 123]. Es wird ein föderiertes Schema unabhängig von den Exportschemata der lokalen Datenquellen entworfen und anschließend werden

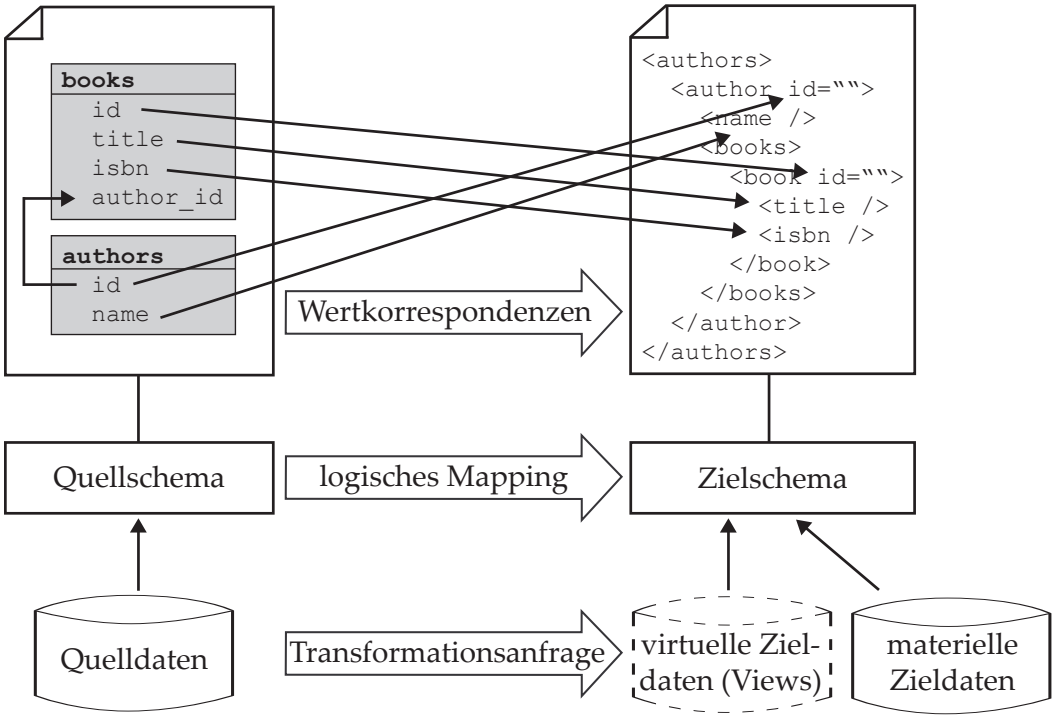

Abbildung 2.21: Schema Mapping Prozess im Überblick, nach [Leser / Naumann, 2007, 125]

Transformationsregeln definiert, mit denen Daten aus den Exportschemata in das föderierte Schema abgebildet werden. Dabei kann die Transformation sowohl materiell ausgeführt, d.h. auf das Schema angewandt werden, als auch virtuell bleiben, d.h. das Schema, auf das sich die Transformation bezieht, bleibt unverändert und lediglich die Ergebnisse einer Anfrage werden transformiert. Ein Mapping ist grundsätzlich *gerichtet*, die Transformation kann also nur vom Quellschema zum Zielschema erfolgen und mit dem gleichen Mapping nicht in die entgegengesetzte Richtung vollzogen werden [Leser / Naumann, 2007, 125].

Abbildung 2.21 zeigt einen Überblick über den Ablauf des Schema Mapping. Ausgangspunkt sind das Quellschema, im Falle der FDBMS das lokale Exportschema einer Datenquelle, und das Zielschema, in diesem Fall das unabhängig erstellte globale föderierte Schema. Zunächst werden wiederum Wertkorrespondenzen zwischen Quell- und Zielschema ermittelt und festgehalten. Dies geschieht, wie in der Abbildung auch, vorzugsweise grafisch, die Wertkorrespondenzen werden als Pfeile zwischen den jeweiligen Attributen gekennzeichnet. Aus den grafischen Korrespondenzen können geeignete Programme ein logisches Mapping, eine Überstzeung der Grafik in Anweisungen, ableiten. Das logische Mapping enthält dann alle Transformationsregeln und dient als

Grundlage für die automatische Erzeugung der Transformationsanfrage. Dies ist eine Anweisung, welche die Quelldaten mit Hilfe der Transformationsregeln in das Zielschema überführt. Wie bereits erwähnt kann diese Transformation materiell sein, die Daten werden in diesem Fall wirklich in die Zieldatenquelle übertragen. Oder die Transformation ist virtuell. In diesem Fall wird in der Zieldatenquelle durch eine *Sichtdefintion* (View) lediglich definiert, wie sie später die Daten „sehen“ soll.

Die Einsatzmöglichkeiten des Schema Mappings sind nicht nur auf FDBMS beschränkt. Die Methode kann vielmehr überall dort angewendet werden, wo Daten von einem Schema in ein anderes transformiert werden sollen, insbesondere auch bei der Migration und Wartung von IT-Systemen [Leser / Naumann, 2007, 123].

2.7.4 ETL-Tools

ETL steht für „Extract Tansform Load“, und bezeichnet den Vorgang, dass Daten aus einer oder mehreren verschiedenen Datenquellen ausgelesen, anschließend zusammengefasst, umgewandelt, transformiert oder korrigiert, und schließlich in ein anderes Datenhaltungssystem überführt werden. Diese Technik wird vor allem für Systeme angewendet, die ihre Funktion aus der Analyse und Gegenüberstellung von gesammelten Daten anderer Systeme ziehen, also für Data Warehouses und Business Intelligence Anwendungen. Für Data Warehoses ist ETL der wichtigste und gleichzeitig aufwändigste Arbeitsschritt. [Bange u. a., 2003, 28]

Das liegt vor allem daran, dass die die syntaktische und semantische Schema-Heterogenität der einzelnen Datenquellen überwunden werden muss, um sie effektiv zusammenführen zu können. Die Homogenisierung der Datenquellen kann damit nicht automatisch erfolgen, sondern es müssen für jeden Einzelfall von einen menschlichen Experten Regeln definiert werden, nach denen die Daten zusammengeführt werden sollen. Für diesen Zweck werden ETL-Tools verwendet.

2.7.4.1 Funktionsweise

Die meisten ETL-Tools funktionieren nach sehr ähnlichen Prinzipien. Das Hauptarbeitsmittel ist ein grafischer Editor, in dem durch die Verknüpfung

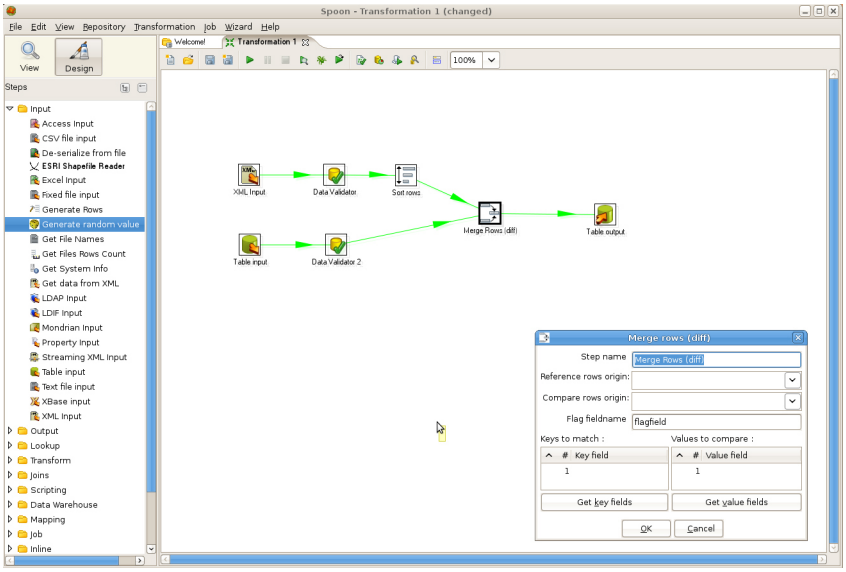

Abbildung 2.22: Grafischer Editor eines ETL-Tools, das Bild zeigt die Oberfläche von *Pentaho Data Integration*

von verschiedenen Objekten – die Datenquellen, Transformations- und Validierungoperationen u.ä. darstellen – ein grafisches Abbild des zu schaffenden Datenfluss entsteht. Die Abbildung 2.22 zeigt das Aussehen eines solchen Editors im OpenSource-ETL-Programm *Pantaho Data Integration*[4]. Zu jedem Objekt steht eine Vielzahl von Einstellungsmöglichkeiten zur Verfügung, die dessen Verhalten in der gewünschten Art beeinflussen. Ist ein solches Datenflussdiagramm fertiggestellt, kann es gespeichert und anschließend komfortabel aufgerufen werden, wobei es alle definierten Arbeitsschritte ausführt. Es kann solange weiterverwendet werden, wie die Datenschemata aller Quelldateien und die definierten Umformungsregeln unverändert bleiben. Die meisten ETL-Tools bieten außerdem die Möglichkeit, den Datenfluss statt über einen grafischen Editor über eine (meist proprietäre) Skriptsprache zu beschreiben.

Ein Datenflussdiagramm für eine ETL-Operation enthält in der Regel die folgenden Ablaufschritte [vgl. Bange u. a., 2003, 28ff]:

Selektion: Zunächst werden die zu verwendenden Datenquellen angegeben. Alle Datenquellen, die verwendet werden sollen, müssen im lokalen System verfügbar sein. Datenquellen können z.B. Datenbanken, Web Services, API-Schnittstellen oder Austauschdateien sein. Viele ETL-Tools unterstützen außerdem das *Change Data Capture*, d.h. sie verwenden nur die Daten, die sich seit dem letzten Aufruf der Datenquelle verändert haben, oder die neu dazugekommen sind.

[4] http://kettle.pentaho.org/, letzter Aufruf am 15.05.2009

Anbindung: Es müssen eventuell Angaben darüber gemacht werden, wie die gewünschten Datenquellen ansprechbar sind. Dazu gehört neben der Angabe von Adresse und Schnittstelle z.B. auch die Angabe von Benutzerdaten für einen Datenbankzugriff.

Überführung: Die Daten werden in diesem Schritt aus der jeweiligen Datenquelle ausgelesen und in den Arbeitsspeicher zur weiteren Verwendung durch das ETL-Tool geschrieben. Änderungen, die nach diesem Zeitpunkt in der Datenquelle geschehen, können also in der aktuellen Operation nicht berücksichtigt werden. Bei der Übertragung muss darauf geachtet werden, dass die Datenzeichensätze korekt sind. Für umfangreiche Daten bietet sich außerdem eine komprimierte Überführung an, um bei dieser Operation Zeit zu sparen.

Filterung und Reinigung: Ausgehend von Vergleichen der Daten der einzelnen Datenquellen werden nun Daten, die aufgrund konkreter Einzelfallregeln nicht weiter verarbeitet werden können, entfernt. Die verbleibenden Daten werden validiert und gereinigt, und weiteren Prozeduren zur Optimierung der Datenqualität unterzogen.

Harmonisierung: Nun erfolgt die Zusammenfassung und Transformation der gereinigten Daten in ein neues Datenschema, das dem des Ladeziels der Operation entspricht. Hier erfolgen wenn nötig auch Umwandlungen von Datentypen, von Datenkodierungen u.ä.

Verdichtung und Anreicherung: In diesem Schritt werden die Daten um Attribute ergänzt, die zwar nicht in den Datenquellen enthalten waren, die aber aus den Daten abgeleitet werden können und entweder vom Ladeziel benötigt werden, oder zu aufwändig sind um zur Laufzeit von diesem erzeugt zu werden. Bei Data Warehouses umfasst dies in der Regel verschiedene betriebliche Kennzahlen, die sich nach komplexen Formeln aus Datenattributen berechnen, oder die Einbindung weiterer externer Daten, durch deren Kombination oftmals neue Erkenntnisse gewonnen werden können.

Laden: Abschließend werden die transformierten Daten in das Ladeziel übertragen. Man unterscheidet hier zwischen dem ersten Laden (Initial Load) und dem oftmals wesentlich kompakteren späteren Ergänzen neuer oder geänderter Werte (Delta Update). Während des Ladevorgangs ist das Ladeziel unter Umständen nicht benutzbar, da sich inhaltliche Dateninkonsistenzen während der Ausführung der Transaktion ergeben können. Das Ladeziel sollte deshalb in der Zeit der Transaktion nicht verwendet werden.

Unter bestimmten Umständen lassen sich ETL-Tools auch als Integrationswerkzeuge anderer Art verwenden. So kann z.B. durch die Definition einer

lokalen Austauschdatei als Quelle und der Datenbank der Anwendung als Ladeziel komfortabel eine Integrationslösung mit einer enfernten Anwendung realisiert werden, die ihre Daten in Form einer Austauschdatei über Syncronisationsdienste wie z.B. `rsync` (vgl. Abschnitt 4.2, S. 104) auf das lokale System überträgt.

3 Der konkrete Integrationsfall

In diesem Kapitel wird unter Verwendung der in Kapitel 2 dargestellten Grundlagen ein konkreter Integrationsfall vorgestellt und untersucht, sowie die Voraussetzungen und Anforderungen für die Integrationslösung fixiert.

Zunächst wird das Integrationsszenario vorgestellt und in die in Kapitel 2 der Arbeit beschriebenen Integrationsformen eingeordnet. Es folgt die Vorstellung der beteiligten Integrationskomponenten und eine Darstellung und Analyse ihrer Datenschnittstellen. Anschließend werden die konkreten Anforderungen des Integrationsszenarios herausgearbeitet, die die Natur der Integrationslösung fundamental bestimmen. Den Abschluss des Kapitels bildet eine Beschreibung der prinzipiellen Gestalt und Funktion der umzusetzenden Integrationslösung.

3.1 Integrationsszenario

Viele Unternehmen nutzen zur Abbildung und Ausführung ihrer Geschäftsprozesse ein Warenwirtschaftssystem (WaWi). Dort werden Daten zu Produkten, Lagerbeständen, Transportvorgängen, Verkäufen, Bestellungen usw. erfasst und verwaltet. Die Daten, die in der WaWi verwaltet werden, sind oftmals sehr umfangreich und von existenziellem Wert für das Unternehmen.

Immer mehr Unternehmen sind im Bereich des E-Commerce tätig, betreiben einen Online-Shop, in dem sie ihre Produkte über den Vertriebskanal Internet zum Verkauf anbieten. Auch in einem Online-Shop werden Daten erfasst und verwaltet. Das umfasst u.a. Produktdaten, Lagerbestandsdaten oder Bestellungsdaten. Diese entsprechen immer zumindest teilweise den Daten in der WaWi, sofern sie dieselben realen Objekte und Fakten repräsentieren. Bspw. werden Informationen zu einem Produkt, wie dessen Name oder Preis, in der WaWi gespeichert sein. Wenn das Produkt nun im Online-Shop angeboten werden soll, so müssen diese Informationen auch im dort verfügbar sein, also in der Datenbank das Shopsystems angelegt werden. Im umgekehrten Fall müssen z.B. Bestellungen aus dem Online-Shop in die WaWi übernommen werden, damit sie im Unternehmen bearbeitet und ausgeführt werden können. Das manuelle oder halbmanuelle Übertragen der Informationen zwischen den beiden Systemen ist zwar prinzipiell möglich, aber aufgrund des hohen Aufwands und der Fehleranfälligkeit in der Realität nicht praktikabel.

Das diesem Kapitel zugrunde liegende Integrationsszenario besteht daher in der Schaffung einer *Verbindung zur automatisierten Übertragung von Daten zwischen den Komponenten Online-Shop und Warenwirtschaft*. Die Datenbestände in beiden Systemen sollen möglichst vollständig übereinstimmen und

eine Datenänderung bzw. Datenerfassung soll vom einen in das andere System übertragen werden, die beiden Komponenten sollen *synchronisiert* werden. Eine solche Verbindung muss in der Lage sein, Konflikte in den zu übertragenden Daten zu erkennen und zu entscheiden, wie die betreffenden Daten zu behandeln sind. Wenn z.B. Adressdaten von Kunden in den Online-Shop importiert werden sollen, und bereits abweichende Adressen für einen der Kunden gespeichert sind, muss entschieden werden, ob die Adressen im Shop beibehalten oder überschrieben werden. Diese Entscheidung hängt in starkem Maße von der konkreten Ausprägung und Herkunft der Daten im Unternehmen ab. Wenn das Unternehmen die Kundendaten weitesgehend selbst erfasst und pflegt, so können die abweichenden Adressen wahrscheinlich überschrieben werden. Wenn die Kunden das Unternehmens ihre Adressen jedoch hauptsächlich selbst angeben, z.B. mit ihrer Anmeldung im Online-Shop, so sollen diese Adressen wahrscheinlich nicht überschrieben werden. Die Entscheidung für eine der Vorgehensweisen ist einzelfallabhängig zu treffen. Die zu schaffende Integrationslösung muss daher beide Fälle gleichermaßen ermöglichen. Die generelle Hauptrichtung der Anpassungen sollte allerdings von der WaWi ausgehend hin zum Shop erfolgen. Die WaWi verfügt in der Regel über weitaus mehr Daten und unternehmerische Spezialfunktionen und ist außerdem fast immer schon vor dem Online-Shop vorhanden. Überdies sind die Mitarbeiter in der Regel in das System der Warenwirtschaft besser eingearbeitet, können ihre Aufgaben hier also schneller und mit weniger Problemen erledigen, als in einem neu zu erlernenden Shopsystem. Und schließlich bietet die WaWi mit ihrem desktopbasierten Client einen performanteren Workflow, als ein browserbasiertes Verwaltungstool das kann. Die Warenwirtschaft ist daher als das führende System anzusehen, der Shop wird weitgehend von der WaWi geführt. Bei Berücksichtigung aller dafür relevanten Daten ist es theoretisch sogar möglich, den Online-Shop vollständig automatisiert über die bereitgestellten Daten der WaWi zu verwalten. Hierbei muss jedoch einschränkend erwähnt werden, dass in der Praxis fast immer manuelle Anpassungen und Pflegemaßnahmen am Online-Shop nötig sind, um eine optimale Präsentation der Produkte zu gewährleisten. Gerade bei der Ausnutzung aller Möglichkeiten, die Magento bietet, ist trotz der Integrationslösung noch viel Handarbeit nötig. Die Grundfunktionalitäten des Tagesgeschäfts können von der Automatisierung aber abgedeckt werden.

Die Natur der im Online-Shop angebotenen Produkte macht eine Bereitstellung aller Daten in „Echtzeit“ in aller Regel nicht erforderlich[1]. Der Grund

[1] Mit Echtzeit ist hier eine Zeitspanne gemeint, die klein genug ist, dass Änderungen in einem System für die menschliche Wahrnehmnung fast sofort im anderen System ankommen, also nicht im Sinne von Gleichzeitigkeit.

Abbildung 3.1: schematische Darstellung des Integrationsszenarios

dafür ist, dass sich die Daten in einem Online-Shop nicht so häufig ändern, dass eine permanente Überprüfung auf Änderungen nötig ist. Weiterhin sind auch die ausgelösten Geschäftsvorgänge (Bestellungen) in der Regel nicht derart zeitkritisch, dass Daten in Echtzeit benötigt werden. Es soll für das Integrationsszenario daher genügen, wenn die Daten in festgelegten Zeitintervallen synchronisiert werden. Zur besseren Abstufung sollen jedoch verschiedene Zeitintervalle für verschiedene Datenoperationen gewählt werden können. Bspw. sollten die Informationen über neue Bestellungen häufiger aktualisiert werden (z.B. alle 30 Minuten), als als etwa die Produktbeschreibungen, bei denen eine Aktualisierung täglich oder auch nur bei Bedarf ausreicht.

Die Integrationslösung soll darüber hinaus in der Lage sein, prinzipiell verschiedene WaWi's an den Online-Shop „anzukoppeln". Es soll später möglich sein, die WaWi einfach und mit wenig Konfigurationsaufwand auszutauschen, um die Lösung für verschiedene Integrationsprojekte mit unterschiedlichen WaWi-Komponenten nutzbar zu machen. Die Shop-Komponente wird dagegen relativ eng mit der Integrationslösung verbunden sein, weil hierfür die eingesetzte konkrete Software, *Magento Commerce*, bereits feststeht[2].

3.2 Umfang der zu integrierenden Daten

Der Zweck der Integrationslösung ist es, den Online-Shop weitgehend automatisch zu pflegen und Daten zwischen den Komponenten zu synchronisieren. Der Umfang der zu berücksichtigenden Datenobjekte richtet sich folglich nach der Schnittmenge der Datenattribute, die von beiden Komponenten interpretiert werden. Eine Information aus der Warenwirtschaft, die der Online-Shop nicht verarbeiten kann, ist für die Integrationslösung nicht verwendbar. Umgekehrt ist auch eine Information aus dem Online-Shop, die von der WaWi nicht verarbeitet werden kann, für die Integration nutzlos. Es sind also diejenigen Informationen von Interesse, die von beiden Systemen verarbeitet werden können. Diese wiederum sollen so umfänglich wie möglich sein, um die Potentiale der Integrationslösung optimal auszunutzen.

[2] Magento Commerce, vgl. Abschnitt 3.3.1, S. 64

In einem Online-Shop sind stets drei grundlegende Datenobjekte von Interesse: *Kundendaten*, *Bestellungsdaten* und *Katalogdaten*. Kundendaten umfassen alle Informationen zu den Kunden, die im Online-Shop einkaufen: Adress- und Kontaktdaten, Zahlungsinformationen, aber auch ihre Einkaufsgeschichte und statistische Angaben zur Nutzung des Shops. Die Bestelldaten umfassen alle Informationen zu einer Bestellung im Shop: Welcher Kunde, welche Lieferadresse, welche Produkte, usw. Die Katalogdaten schließlich umfassen das Produktangebot und die Angebotsstruktur des Shops. Jedes dieser Objekte enthält eine Menge von Attributen, die ihrerseits zum Teil wiederum Datenobjekte darstellen. Eine Bestellung enthält z.B. die bestellten Produkte als Unterobjekte. Zur Abbildung der Hierarchie der in einem Datenobjekt enthaltenen Attribute und Unterobjekte eignet sich die Verschachtelung. Die Datenschnittstellen beider Integrationskomponenten nutzen diese Abbildungsvorschrift.

Eine Besonderheit des in dieser Arbeit verwendeten Online-Shops Magento Commerce ist, dass für einige Datenobjekte Attribute individuell angelegt werden können. Es ist damit möglich, dass ein Objekt zu einem Zeitpunkt A andere Attribute besitzt als zu einem Zeitpunkt B. In der Regel ändern sich Attribute von Datenobjekten aber selten. Dennoch muss es die Integrationslösung ermöglichen, Abildungsvorschriften für solche neuen order geänderten Attribute aufzunehmen. Das geschieht über sog. *Mappingdateien*. In Mappingdateien werden die Datenattributschlüssel der konkreten Datenobjekte mit Systembezeichnern verknüpft, die eine Übertragung in ein anderes Datenschema ermöglichen (siehe Abschnitt 4.6, S. 117). Durch die Ergänzung neuer Systemschlüssel können somit neue Datenattribute für die Integration benutzt werden.

Im Rahmen dieser Arbeit werden für die umgesetzte Integrationslösung längst nicht alle Datenattribute verwendet, die die Integrationskomponenten verarbeiten können. Dies geschieht aus dem Grund heraus, dass die Integrationslösung keine produktiv einsetzbare Lösung darstellen soll, sondern lediglich als Umsetzungsbeispiel konzipiert ist. Es werden stattdessen nur die wesentlichsten Grunddaten verwendet. Eine Aufstellung aller in der Integrationslösung verwendeten Datenobjekte und Attribute ist in Anhang A.1, Seite XVI einzusehen. Weitere individuelle Attribute können je nach konkretem Praxisfall zusätzlich verwendet werden.

3.3 Vorstellung der Integrationskomponenten

3.3.1 Magento Commerce

Als zu integrierendes Online-Shopsystem wurde *Magento Commerce*[3] vom amerikanischen Softwarehersteller Varien gewählt. Magento ist eine OpenSource-Software, ist also kostenfrei zu erhalten und durch offenliegenden Quellcode für eigene kommerzielle Einsatzzwecke einsetz- und anpassbar. Magento ist unter der Open Software Licence (OSL) 3.0 lizenziert. Das bedeutet, dass jeder den Quellcode für seine Zwecke verwenden und anpassen kann, dass Änderungen und Erweiterungen aber ebenfalls unter die OSL-Lizenz fallen und der Community zurückgegeben (verfügbar gemacht) werden müssen [Zenner, 2009, 12]. Eine große Community im Internet entwickelt das System in Form von Modulen weiter und gibt Hilfestellung bei Problemen, und auch Varien pflegt und aktualisiert Magento mit viel Aufwand. Das System ist objektorientiert in → PHP geschrieben und basiert auf dem → Zend Framework. Durch den modularen Aufbau und die klare Systemarchitektur ist die Software sehr gut erweiterbar und anpassungsfähig und kann dennoch leicht gepflegt und jederzeit aktualisiert werden [vgl. Willkommer, 2008, 26]. Es bietet viele professionelle Features und Tools, und ist insgesamt auf die Nutzung für umfangreiche und stark frequentierte Online-Shops mittlerer und größerer Unternehmen zugeschnitten.

Magento wurde am 31. März 2008 in der ersten stabilen Version 1.0 (*Production-Release*) veröffentlicht. Nach Aussage von Varien hat Magento die am schnellsten wachsende Community aller OpenSource-Shopsysteme. Bereits vor der Veröffentlichung der ersten stabilen Version wurde es über 200.000 Mal heruntergeladen [Stockbauer, 2008, 32]; bis zum gegenwärtigen Zeitpunkt wurde soagr die Zahl von 750.000 Downloads überschritten. Im Jahr 2008 wurde Magento mit dem „SourceForge.net Community Choice Award“ für das beste neue OpenSource-Projekt des Jahres ausgezeichnet. Zum 31.03.2009, dem einjährigen Jubiläum des ersten Production-Release, wurde die Version 1.3 veröffentlicht, die wie die vorherigen Releases zahlreiche Verbesserungen, Neuerungen und Bugfixes enthält. Das Entwicklungstempo von Varien und das Engagement der Community ermöglichen Magento derzeit eine sehr agile Weiterentwicklung. Zur Zeit noch in der Early-Adopter-Phase befindlich, deutet vieles darauf hin, dass sich Magento auf mittlere Sicht zu einem der bedeutendsten OpenSource-Ecommerce-Systeme am Markt entwickeln wird [Techdivision, 2009, 8]. Forrester Research bezeichnet es schon jetzt als „Emerging

[3] http://www.magentocommerce.com, letzter Aufruf am 31.03.2009

Abbildung 3.2: Startseite des Magento-Demoshop.
Screenshot von http://demo.magentocommerce.com

Player to Watch“ [Zenner, 2009, 1].

Das Produktkonzept

Das Konzept der Abbildung von Produkten in Magento unterscheidet sich deutlich von anderen Shopsystemen. Magento kennt neben normalen Produkten, den sog. *Simple Products*, noch eine Reihe anderer Produkttypen. Viele dieser Produkttypen dienen der übersichtlicheren Anordnung der Produkte im Shop oder stellen alternative Verkaufskanäle zur Verfügung. Einige von ihnen sollen im Folgenden in Anlehnung an die Ausführungen in [Zenner, 2009, 128] kurz vorgestellt werden.

Als Simple Products werden Produkte bezeichnet, die in nur einer *Produktausprägung* in einem Shop vorliegen können, wobei unter Produktausprägung die Gesamtheit aller relevanten Eigenschaften des Produkts zu verstehen ist. Ein Buch ist ein gutes Beispiel für ein solches Simple Product. Ein einzelnes Buch hat in der Regel immer nur eine Produktausprägung, d.h. einen Preis, eine Gestaltung des Einbands, eine Papierqualität, usw. Simple Products sind die Basis für die weiteren Produkttypen, die Magento bietet.

Ein *Configurable Product* ist eine Zusammenfassung von mehreren Simple Products zu einem Produkt, das in mehreren Produktausprägungen vorliegen kann. Ein Beispiel für ein Configurable Product ist ein T-Shirt, das mit einem bestimmten Motiv bedruckt ist. Das T-Shirt kann in verschiedenen

Farben, Schnitten und Größen im Shop verfügbar sein, das Motiv stellt jedoch das entscheidenede Produktmerkmal dar. Die Kombination der Merkmalsausprägungen – Farbe, Schnitt, Größe – führt zu einer stattlichen Anzahl an Produktausprägungen für ein und dasselbe Motiv. Anstatt nun alle diese Produktausprägungen im Shop anzuzeigen, wird nur ein gruppiertes Produkt angezeigt. Das T-Shirt-Motiv taucht also im Shop nur einmal auf, und der Nutzer kann sich die Ausprägung der einzelnen Merkmale (Farbe, Schnitt, Größe) selbst zusammenstellen. In der Datenhaltung des Shops existiert aber jede Produktausprägung als eigenständiges Simple Product. Die wesentlichen Vorteile solcher Configurable Products sind [vgl. Stockbauer, 2008, 32]:

- *übersichtlicher Shop*:
  Das Produkt wird im Shop nur einmal angezeigt; verschiedene Ausprägungen überschwemmen nicht den Shop mit einer Vielzahl fast identischer Produkte.
- *Benutzerinteraktion*:
  Der Benutzer kann Merkmalsausprägungen nach seinen Wünschen festlegen und erhält so das Gefühl, sein individuelles Produkt selbst zu erstellen.
- *Verwaltung auf konkrete Produkte bezogen*:
  Obwohl der Kunde das Gefühl hat, sein Produkt selbst zu kreieren, sind die einzelnen Produkte als konkrete Simple Products im Shop gespeichert. Somit können für jede Ausprägung des Configurable Product genaue Aussagen über Verfübarkeit, Anzahl der Produkte im Lager, Verkaufszahlen, Sonderpromotionen usw. getroffen werden.

Ein *Grouped Product* ist ebenfalls eine Zusammenfassung von mehreren Simple Products auf einer Detailseite im Shop. Im Unterschied zum Configurable Product kann ein Benutzer auf dieser Seite aber mehrere verschiedene enthaltene Simple Products bestellen. Es stellt also eine Liste von verwandten Simple Products dar, mit welcher der Benutzer mehrere Produkte auf einmal in seinen Warenkorb legen kann. Grouped Products eignen sich vor allem für Produkte, die sich untereinander in nur einem Merkmal unterscheiden, und von denen ein Benutzer verschiedene Ausprägungen gleichzeitig benötigen kann. Ein Beispiel für ein Grouped Product sind Batterien. Verschiedene Batteriegrößen können zu einem Grouped Product zusammengefügt werden. Der Benutzer kann dann aus einer Liste auf ein und derselben Produktdetailseite die gewünschte Menge von jeder Größe bestellen.

Mit dem *Downloadable Product* können auch Produkte über Magento verkauft werden, die im Anschluss an den Kauf direkt heruntergeladen werden können.

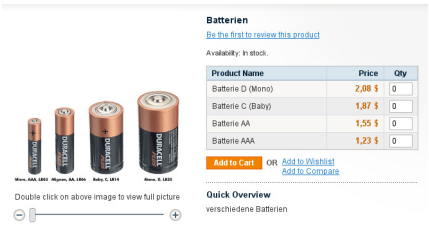

Abbildung 3.3: Beispiel für ein Grouped Product in Magento

Das umfasst z.B. Software oder Musik. Nach dem Abschluss der Bestellung sind Downloadable Products in einem eigenen Bereich des Shops direkt abrufbar. Im Gegensatz zum Simple Product wird also bei einer Bestellung kein Versand fällig. Dieser Produkttyp ist der Neueste in Magento. Er wurde in Version 1.2 eingeführt.

Das *Bundle Product* ist eine feste Kollektion von Simple Products, die als ein Produkt gekauft wird. So kann etwa ein Shop, der Computerkompontenten verkauft, in einem Bundle Product einen kompletten Computer anbieten, der eigentlich aus einzelnen Komponenten in Form von Simple Products besteht. Bei einem Bundle kann der Käufer nicht zwischen den enthaltenen Simple Products auswählen, sondern muss das komplette Bundle kaufen. Dafür ist ein Bundle meist billiger als die einzelnen Komponenten zusammengerechnet.

Der letzte Produkttyp ist das *Virtual Product*. Es funktioniert prinzipiell wie ein Simple Product und unterscheidet sich von diesem nur dadurch, dass bei der Bestellung kein Versand nötig wird, da das Produkt nicht materiell ist. Bei einem Virtual Product kann es sich z.B. um erweiterte Serviceleistungen oder um eine Dienstleistung handeln.

Features für Benutzer

Eines der wichtigsten Features von Magento Commerce ist seine durchgehend auf optimale →Usability (Gebrauchstauglichkeit) ausgerichtete Gestaltung. Vom ersten Moment der Benutzung der Standardinstallation an wird klar, dass hier das Hauptaugenmerk der Entwickler lag. Der Shop ist intuitiv gestaltet, alle Funktionalitäten sind klar erkennbar und dienen einem konkreten Nutzerbedürfnis. Viele Funktionen sind vom Partizipationsgedanken des *Web 2.0* inspiriert und bieten dem Benutzer ein Einkaufs- und Surferlebnis mit zeitgemäßen und direkten Interaktionsmöglichkeiten. Dazu gehören Bewertungs-, Kommentar- und Schlagwortfunktionen ebenso, wie innovative Such- und Produktfilterfunktionen und viele weitere.

Magento bietet den Benutzern auch einen *Single Page Checkout*. Das bedeutet, dass alle Angaben, die für eine Bestellung nötig sind, auf einer Seite erfasst werden. Der Benutzer wird besser durch den Vorgang der Bestellungsabwicklung geführt und muss sich durch weniger Seiten klicken, was mit zu einer höheren Konversionsrate[4] beiträgt. Viele notwendige Validierungsaufgaben (Überprüfen von Kreditkarten- und Adressdaten etc.) werden mit AJAX-Requests im Hintergrund getätigt und liefern bei Bedarf sofort Rückmeldungen, bevor der Nutzer auf „Bestellung senden“ klickt. Der gesamte Vorgang ist darauf optimiert, dem Besteller so wenig Anlass wie möglich zu bieten, aufgrund von Konfusion, Frustration oder Unverständnis den Bestellvorgang abzubrechen.

Bei Bedarf nach weiteren Features bietet Varien mit *Magento Connect*[5] eine umfangreiche Sammlung von Erweiterungen an, die über den Verwaltungsbereich von Magento komfortabel installiert werden können. Diese Erweiterungen werden zum Großteil von der Community entwickelt und stellen eine Vielzahl von speziellen Funktionalitäten bereit.

Features für Shop-Betreiber

Eines der wichtigsten Features für Betreiber von Onlineshops sind die umfangreichen Funktionen für *Marketing* und *Kundenpflege*, die Magento mitbringt. Diese reichen von Rabattstaffeln, gestaffelten Preisen für verschiedene Kundengruppen wie Wiederverkäufer oder Großabnehmer, über Cupon- und Promotionaktionen, Newsletterverwaltung, bis hin zu Cross-Sells und Up-Sells, also der Anzeige von anderen bzw. teureren Produkten, die ein Interessent bei einem Produkt ebenfalls zum Kauf in Erwägung ziehen soll. Über den Verwaltungsbereich lassen sich diese Werkzeuge alle einfach einsetzen und weitgehend automatisieren. Zum Beispiel können zeitlich beschränkte Promotionaktionen mit wenigen Klicks erzeugt und aktiviert werden, sodass sie sofort im Shop, z.B. auf der Startseite sichtbar werden.

Magento kann mehrere voneinander getrennte Shops betreuen und verwalten. Diese können alle über einen Verwaltungsbereich gepflegt werden. Es können mehrere Versionen eines Shops vorhanden sein, die sich Produkte teilen, z.B. Versionen in verschiedenen Sprachen mit unterschiedlichen Layouts aber zum

[4] Die Konversionsrate ist ein statistischer Messwert, der das Verhältnis von Käufern zu Kaufinteressenten ausdrückt. Je höher die Konversionsrate umso größer ist der Anteil der Benutzer die bei einem Besuch des Shops auch etwas kaufen.

[5] http://www.magentocommerce.com/magento-connect, letzter Aufruf am 13.04.2009

Teil identischen Produkten. Shops können aber auch komplett voneinander unabhängig sein und auch über getrennte Lagerbestände verfügen. Diese Funktionalitäten werden mit dem Begriff *Multistore-Fähigkeiten* zusammengefasst.

Auf der systemtechnischen Seite verfolgt Magento den Ansatz, im besten Sinne des Wortes eine *Enterprise-Anwendung* zu sein, ein System also, das den hohen Ansprüchen von professionellen Anwendern genügt, und komplexe und stark frequentierte Online-Angebote in hoher Qualität und mit komfortablen Funktionen bedienen kann. Da Magento auf dem → Zend Framework basiert, modular aufgebaut und flexibel erweiterbar ist, verspricht es eine sehr gute → Skalierbarkeit zu erreichen und ist so auch für überaus umfangreiche Shops geeignet. Gleichzeitig wird durch den Framework-Ansatz sichergestellt, dass der Kern der Anwendung unverändert bleibt und dauerhaft stabil und updatefähig gehalten wird.

Hohe Anforderungen an System und Betreiber

Die Funktionen und Möglichkeiten, die Magento bietet, gehen weit über den Funktionsumfang bisheriger OpenSource-Systeme hinaus. Diese Leistung hat jedoch ihren Preis. Dieser offenbart sich zum einen im Umfang der benötigten *Ressourcen*, um einen Shop mit Magento zu betreiben und zum anderen im *Pflegeaufwand*, der nötig ist, um die vorhandenen Möglichkeiten effektiv zu nutzen.

Eines der vordergründigsten Merkmale eines guten Online-Shop ist, dass er performant reagiert und sich in der Bedienung nicht zäh anfühlt. Seit der Veröffentlichung der Version 1.0 Ende März 2008 ist die Performance, insbesondere die Geschwindigkeit des Seitenaufbaus, in Magento-Shops ein vieldiskutiertes Thema in den Blogs und Foren der Magento-Community. Einigkeit besteht darüber, dass ein Magento-Shop eine deutlich üppiger ausgestatte Serverhardware benötigt, als ein vergleichbarer Shop, der auf einem anderen OpenSource-Shop, wie bspw. xt:Commerce basiert. Bereits für kleinere Shops mit einigen hundert Produkten und Kunden pro Tag sollte ein dedizierter Server verwendet werden. Shared-Hosting Pakete sind nur in sehr begrenztem Maße für Magento-Shops geeignet. Ebenfalls von enormer Bedeutung ist der Einsatz von → Caching zur Performanceverbesserung. Für einen performanten Shop müssen sowohl die in Magento integrierten Cachingfunktionen optimal verwendet werden, als auch zusätzliche Cachingwerkzeuge für das Kompilieren von PHP, sog. *Opcode Caches* wie etwa APC oder eAccelerator, und für das Optimieren von Datenbankoperationen kombiniert eingesetzt werden. Dies erfordert einiges an Kenntnis

und Einrichtungsaufwand und ist meist nur auf einem dedizierten Server überhaupt möglich.

Viele der neuen Funktionen verlangen nach Handarbeit durch den Shopbetreiber. Um z.B. Cross-Sells und Up-Sells, verwandte Produkte oder auch Kundenrezensionen und Tags effektiv und zur Steigerung des eigenen Umsatzes auszunutzen, bedarf es eines nicht unerheblichen Zeitaufwands, da diese Anpassungen individuell und nicht automatisierbar sind. Ein solcher Shop muss in der Regel von einem oder mehreren Redakteuren betreut werden. Je besser der Inhalt und die Pflege des Shops ausfallen, desto mehr Kunden wird das Angebot überzeugen können. Auch diese Investition dürfte eher durch größere Firmen geleistet werden.

benötigte Hard- und Software

Wie bereits dargelegt wurde, benötigt Magento praktisch immer einen dedizierten Server. Um mit genügender Performance zu laufen, sollte dies ein modernes System mit mindestens 2 Prozessorkernen und 2 Gigabyte Arbeitsspeicher sein, für größere Shops sind 4 Prozessorkerne, sowie deutlich mehr Arbeitsspeicher vorzuziehen [Stockbauer, 2008, 34]. Der benötigte Speicherplatz ist sehr einzelfallabhängig und richtet sich nach dem Umfang der abzulegenden Daten, insbesondere von Bildern und evtl. weiteren verwendeten Medien. Die Spanne reicht hier von wenigen bis hin zu mehreren hundert Gigabyte.

Auf der Softwareseite können alle benötigten Komponenten aus kostenfreien und erprobten OpenSource-Anwendungen bestehen. Als Betriebssytem sollte die Serverversion einer Linux-Distribution verwendet und durch einen Systemadministrator optimal und sicherheitsorientiert konfiguriert werden. Aufgrund der langen Lebenszyklen und den oftmals sehr auf Sicherheit getrimmten, verlässlichen Paketen sind Enterprise-Versionen von Debian oder Red Hat-Linux in besonderem Maße geeignet [Zenner, 2009, 17]. Sollen mehr als 4 Gigabyte Arbeitsspeicher verwendet werden, so muss eine entsprechende 64-Bit-Version des Betriebssystem zum Einsatz kommen, die den großen Arbeitsspeicher auch ausnutzen kann [Zenner, 2009, 17]. Darüber hinaus sollte ein Apache 2 Server in Verbindung mit einem aktuellen PHP (derzeit PHP 5.2.9) und MySQL (derzeit MySQL 5.1) verwendet werden. Für PHP wird weiterhin noch eine Reihe von Extensions benötigt, die unter Umständen manuell installiert werden müssen.[6] Ist diese Software vorhanden, kann Magento in Betrieb

[6] Eine vollständige Auflistung der benötigten Extensions und der unterstützten Softwareversionen ist unter http://www.magentocommerce.com/system-requirements abrufbar. Letzter Aufruf am 06.05.2009

genommen werden. Für optimale Performance sollte unbedingt ein PHP Opcode Cache verwendet werden. Empfehlenswert ist außerdem der Einsatz eines Query Cache für MySQL-Anfragen [Zenner, 2009, 18]. Dieser arbeitet ähnlich wie ein PHP Opcode Cache und speichert bereits kompilierte Anfragen, die dann bei einem erneuten Aufruf deutlich schneller zur Verfügung stehen. Des weiteren sollten alle Komponenten (Apache, PHP, MySQL) stets in der aktuellsten stabilen Version verwendet werden, da jedes neue Release in der Regel Sicherheistlücken schließt und die Performance verbessert. Ein weiterer Performancefaktor ist die optimale Konfiguration der Linux-Distribution. [Stockbauer, 2008, 35] spricht von einem möglichen Geschwindigkeitsgewinn von bis zu 100 Prozent gegenüber einem unoptimierten System. An der Anpassung und Optimierung des Systems sollte daher keinesfalls gespart werden.

3.3.2 ALEA Commerce Suite

Als im Rahmen des Integrationsszenarios anzubindende Warenwirtschaft wurde *ALEA Commerce Suite* von ALEA gewählt. ALEA ist eine deutsche Softwarefirma mit Sitz in Jena, die sich auf Software für Versandhäuser spezialisiert hat. Die ALEA Commerce Suite ist eine neu entwickelte *Branchensoftware* für den Versandhandel. Als Branchensoftware wird eine Software bezeichnet, die als Standardlösung für eine Branche entwickelt wurde und alle für das Marktsegment nötigen Standardfunktionen mitbringt. Sie ist dadurch kurzfristig einsetzbar und verfügt in der Regel über brachenspezifisches Expertenwissen. ALEA Commerce Suite deckt als Warenwirtschafts- und Logistiksoftware in der Standardinstallation bereits alle Kernprozesse ab, die in der Versandhandelsbranche auftreten [SoftGuide GmbH & Co. KG, 2009]. Dazu gehören neben den Kernprozessen Angebotsentwicklung, Auftragsabwicklung und Logistik auch unterstützende Supportprozesse, wie bspw. Kundenbetreuung, Lagerhaltung oder Debitorenmanagement, sowie ergänzende Managementprozesse, die z.B. Controlling und Data Warehousing umfassen [ALEA Vertriebspräsentation, 2009, 12].

Multi-Channel-Prozesse

Besondere Bedeutung hat in ALEA Commerce Suite die uneingeschränkte Unterstützung von *Multi-Channel-Prozessen* in allen Geschäftsbereichen des Versandhandels [ALEA Vertriebspräsentation, 2009, 12]. Unter Multi-Channel-Prozessen sind Geschäftsabläufe zu verstehen, deren einzelne Schritte auf verschiedenen Kommunikations- und Geschäftskanälen gleichberechtigt ablaufen

oder ablaufen können, bzw. zwischen diesen wechseln können. Ein Beispiel für einen solchen Multi-Channel-Prozess ist der Bestellvorgang eines Kunden, der sich etwa im Print-Katalog der Firma über ein Produkt informiert, über die Kundenservice-Hotline genauere Fragen beantwortet bekommt und schließlich das Produkt über das Internet bestellt. Teilaufgaben des Geschäftsprozess „Bestellung eines Produkts“ werden also auf verschiedenen Kanälen durchgeführt, es wird dadurch komplexer, den Prozess als Ganzes zu überblicken und abzubilden. Auch nach der Bestellung können viele Prozesse auf verschiedenen oder mehreren Kanälen ablaufen: ob die Lieferung des Produkts per Zustellung oder durch Abholung an einer Packstation erfolgt, ob die Zahlung per Kreditkarte, Vorkasse oder Rechung beglichen wird, usw. Es verlangt vom Unternehmen und von der eingesetzten Software einiges an Flexibilität und Know-How, alle Prozessabschnitte zu einem einheitlichen Multi-Channel-Prozess verbinden, der auch alle anderen Unternehmensbereiche umspannt. Die Lagerlogistik und die Warenbeschaffung etwa sollten auch mit den Multi-Channel-Prozessen Schritt halten können.

Multi-Mandanten-Fähigkeit

Neben der Eignung für Multi-Channel-Prozesse ist ALEA Commerce Suite auch *multi-mandantenfähig*. Mandanten sind als Teilbereiche eines Unternehmens vorstellbar, die zwar weitgehend autonom handeln, die aber in bestimmten Punkten den zentralen Datenbestand verwenden können. Ein Mandant ist z.B. ein Filial-Geschäft eines Unternehmens. Es benutzt einen eigenen Datenstamm für die Lagerhaltung vor Ort, kann aber für die Warenbeschaffung und die Logistik den gemeinsamen Datenstamm des Gesamtunternehmens nutzen. Damit entstehen vielfältige Synergieeffekte zwischen den Mandanten [ALEA Produktbroschüre, 2009, 4]: Im genannten Beispiel kann eine zentralisierte Warenbeschaffung u.a. den Einkaufspreis erheblich senken und eine gemeinsame Logistik deren Auslastung und Effizenz verbessern. Auch eine gemeinsame Nutzung von Kundendaten – z.B. Angaben zur Bonität oder zu Kaufgewohnheiten – kann für das Unternehmen positive Effekte generieren. ALEA Commerce Suite bietet ein flexibles Mandantensystem, in dem sowohl zentralisierte als auch individuelle Geschäftsfelder je nach Bedarf kombiniert werden können [ALEA Produktbroschüre, 2009, 4].

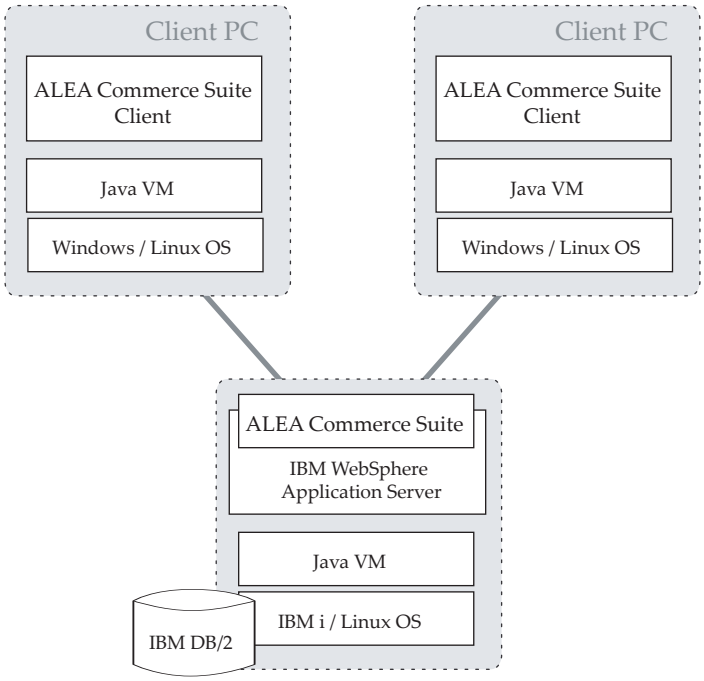

Abbildung 3.4: Schematischer Aufbau der ALEA Commerce Suite , Grafik erstellt nach [ALEA Vertriebspräsentation, 2009, 6]

Technik

Der prinzipielle Aufbau von ALEA Commerce Suite ist in Abbildung3.4 dargestellt. Die Anwendung funktioniert nach dem Client/Server-Prinzip. Auf einem zentralen Anwendungsserver im Unternehmen läuft die Serverseite der Software. Clients können auf beliebig vielen Rechnern installiert werden und nutzen für die durchzuführenden Operationen Funktionalitäten auf dem Server. Die Branchensoftware ist vollständig in Java umgesetzt und verwendet die standardisierte J2EE bzw. Java EE Architektur [SoftGuide GmbH & Co. KG, 2009]. Das macht die Anwendung unabhängig, sowohl vom verwendeten Application Server und vom Betriebssystem, als auch von bestimmten Datenhaltungen. Sie kann mit allen verbreiteten Systemen verwendet werden. Im Bereich der Application Server sind das vor allem IBM WebSphere, SAP NetWeaver oder auch JBoss; bei der Datenhaltung kommt – je nach verwendetem Application Server und Betriebssystem – vor allem IBM DB2 zum Einsatz. Der Application Server stellt dabei der Anwendung bestimmte Dienste auf dem System, wie z.B. Transaktionsverwaltung oder den Zugriff auf eine Datenhaltung, zur Verfügung und funktioniert als Laufzeitumgebung für die Anwendung selbst. Das Betriebssystem der Serverumgebung ist ebenfalls vom Application Server abhängig. Im allgemeinen kommt entweder IBM OS/400 oder eine Linuxdis-

tribution zum Einsatz. Der Client funktioniert derzeit unter Windows- und Linux-Betriebssystemen. Server und Client benötigen desweiteren eine installierte Java → Virtual Machine (VM), die den Java-Bytecode ausführt.

Alle Schnittstellen und Komponenten in Java EE werden in einem offenen Prozess (Java Community Process - JCP) in Zusammenarbeit mit den großen Herstellern und Anwendern entwickelt [Koschel u. a., 2006, 2]. Die Plattform bleibt somit sehr praxisnah und erhält eine Dynamik zur Unterstützung neuer Anforderungen, die anderen Sprachen oft fehlt. Ein wichtiger Grundsatz von Java EE ist die Trennung von Präsentation und Geschäftslogik, sowie die Trennung von fachlicher Logik von der technischen Umsetzung [Bien, 2007, 23]. Somit ist ALEA Commerce Suite transparent in der Umsetzung, einfach zu pflegen und modular erweiterbar, bei geleichzeitiger Erhaltung der Updatefähigkeit der Kernmodule. Es verwendet für die Integration aller Systemkomponenten untereinander einen → Enterprise Service Bus (ESB), der diese vom zu verwendenden Protokoll unabhängig macht, eventuell nötige Datentransformationen vornimmt und weitere Funktionen wie Zugriffskontrollen und Monitoring bietet [ALEA Vertriebspräsentation, 2009, 9]. Schließlich wird der vollständige Quellcode und ein kommentiertes Datenbankmodell mit dem System ausgeliefert, sodass für das Unternehmen alle Möglichkeiten einer individuellen Weiterentwicklung und Erweiterung gegeben sind [ALEA Vertriebspräsentation, 2009, 10].

3.4 Analyse des Integrationsszenarios

Um die Beschaffenheit der Integrationslösung in optimaler Anpassung an das Integrationsszenario bestimmen zu können, ist es zunächst notwendig, dieses genauer zu hinterfragen und Angaben bezüglich der wichtigsten Grundmerkmale zu erhalten. Dabei sollen die Ausführungen und Kategorisierungsansätze aus Kapitel 2, insbes. aus dem Abschnitt 2.5, S. 25 als Grundlage dienen.

3.4.1 Richtung des Integrationsszenarios

Der Zweck, der mit dem vorgestellten Szenario verfolgt wird, besteht darin Unternehmensprozesse entlang der Wertschöpfungskette *Business-to-Customer* zu optimieren. Durch die Automatisierung von Abläufen und Informationsflüssen von Onine-Shop auf der Kundenseite und der WaWi auf der Unternehmensseite können Angebots-, Bestellungs- und Lieferprozesse schneller, effektiver und weniger fehleranfällig gestaltet werden. Sowohl der Kunde als auch das

Unternehmen erfahren durch den Einsatz einen Mehrwert: Der Kundenwunsch kann schneller bearbeitet werden, zudem ist der Kommunikationsweg zwischen Kunde und der zuständigen Stelle im Unternehmen deutlich verkürzt. Der Mehrwert des Unternehmens liegt, neben einer Erhöhung der Kundenzufriedenheit, vor allem in der Reduktion von Arbeitsaufwand bei der Abbildung der Geschäftsvorgänge durch die Automatisierung von Informationsflüssen. Das vorliegende Szenario optimiert also Vorgänge entlang der Wertschöpfungskette und ist damit ein *horizontaler Integrationsansatz* (vgl. Abschnitt 2.5.1.1, S. 25).

3.4.2 Kommunikationsmodell des Integrationsszenarios

Das Integrationsszenario beschreibt zwei Integrationskomponenten, die voneinander autark arbeiten sollen. Die Verfügbarkeit der einen Komponente soll keine direkten Auswirkungen auf die Verfügbarkeit der anderen Komponente haben. Die Kommunikation zwischen den Komponenten muss nicht in „Echtzeit“ erfolgen. Daraus geht hervor, dass ein *asynchrones Kommunikationsmodell* zwischen den Integrationskomponenten vorliegt (vgl. Abschnitt 2.5.3.2, S. 29). Eine Übertragung von Informationen in „Echtzeit“ ist nicht gefordert, die angegebenen Intervalle lassen sich hervorragend durch einen zyklischen Aufruf der Integrationslösung realisieren, welche die zu diesem Zeitpunkt vorhandenen Informationen an die empfangende Integrationskomponente überträgt. Dies bedingt eine Zwischenspeicherung der Informationen innerhalb der Reichweite der Integrationslösung.

Die Anwendungen sind durch das asynchrone Kommunikationsmodell, wie vom Integrationsszenario verlangt, weitesgehend autark voneinander. Sie stellen Informationen zur Verfügung, müssen sich aber nicht darum kümmern, ob diese Informationen an der richtigen Stelle ankommen und ob sie ordnungsgemäß verarbeitet werden. Sie sind also nach der Bereitstellung der Informationen nicht weiter involviert und müssen keine Ressourcen freihalten um auf Bestätigungen zu warten oder eventuell eingehende Fehler zu verarbeiten. Das bedeutet weiterhin auch, dass keine anderen Eingriffe in die Integrationskomponenten selbst erfolgen müssen, als solche, die die Bereitstellung der Informationen in geeigneter Form und in Reichweite der Integrationslösung zum Zweck haben.

3.4.3 Integrationsobjekte des Integrationsszenarios

Als Integrationsobjekte sind im vorliegenden Szenario die Anwendungen zu betrachten. Das bedeutet, dass die Anwendung selbst die zur Nutzung durch den Integrationspartner bestimmten Informationen erzeugt und in einem Zwischenspeicher ablegt. Aus diesem Zwischenspeicher gelangen die Informationen dann zum Integrationspartner. Die Integrationskomponenten verwenden also keinen gemeinsamen Informationsspeicher (z.B. Datenbank), sondern erzeugen die Informationen nach Bedarf selbst. Dies charakterisiert das Integrationsszenario als Fall der *Anwendungsintegration* (vgl. Abschnitt 2.5.2.2, S. 27). Eine Integrationslösung über die Datenhaltungsschicht (Informationsintegration) ist für dieses Szenario nicht anzustreben, da zum Einen die Anwendungen autark bleiben sollen, d.h. auch voneinander unabhängige Datenhaltungsschichten verwenden sollen. Zum Anderen ist ein Online-Shop und dessen Datenhaltung über das Internet erreichbar. Natürlich sollte der Zugang entsprechend gut gesichert sein. Prinzipiell ist die Datenhaltung aber weltweit erreichbar. Die Warenwirschaft eines Unternehmens beinhaltet jedoch in aller Regel auch sensible Daten, die keinesfalls über das Internet zugänglich und damit potentiell manipulierbar sein dürfen. Eine integrierte, über das Internet erreichbare, Datenhaltung scheidet deshalb als Sicherheitsrisiko für die Integrationslösung aus.

Die Tatsache, dass die Integrationskomponenten die zur Integration bestimmten Daten selbst erzeugen, ermöglicht eine feinkörnige Einstellung von Art und Umfang der zu integrierenden Informationen. Die Daten, die im Zwischenspeicher für die Integration bereitgestellt werden, werden allein durch die jeweilige Integrationskomponente bestimmt und erzeugt. Somit bleibt jede Integrationskomponente in ihrem Datenbestand völlig autark. Die anderen beteiligten Komponententen haben keine Möglichkeit, diesen Datenbestand in ungewollter Weise zu manipulieren. Alle Funktionalitäten der Anwendung zur internen Aufbereitung, Konsolidierung und Validierung der Daten können sowohl vor der Bereitstellung als auch vor der Übernahme von Daten genutzt werden, um eine optimale Qualität der bereitgestellten Informationen zu gewährleisten.

3.4.4 Intensität der Integrationslösung

Die Integrationslösung soll eine hohe Intensität der Integrations erreichen. Alle relevanten Informationen und Operationen mit diesen sollen vollautomatisch integriert werden. Der Aufwand für das Einpflegen der Informationen in den Shop soll weitesgehend entfallen, die Integrationslösung soll möglichst

transparent arbeiten. Für den Fehlerfall müssen daher Benachrichtigungs- und Protokollierungsfunktionalitäten in der Integrationslösung implementiert werden, damit ein zeitnahes Eingreifen und die Beseitigung des Fehlers ermöglicht ist.

3.4.5 Zusammenfassung der Grundmerkmale

Aus der Analyse des Integrationsszenarios wurde eine Liste von Grundanforderungen an die Integrationslösung abgeleitet. Diese Grundanforderungen bestimmen wesentlich die Gestaltung und Umsetzung der Integrationslösung. Sie sollen im Folgenden noch einmal kurz zusammengefasst werden.

- *Flexible Erweiterbarkeit für andere WaWi's*:
  Der Aufbau der Integrationslösung soll es ermöglichen, verschiedene WaWi's als Datenquelle für Magento nutzen zu können. Diese sollen mit möglichst geringem Aufwand anstelle der in dieser Arbeit verwendeten ALEA Commerce Suite eingesetzt werden können.
- *Keine Echtzeitdaten*:
  Die Daten müssen nicht in Echtzeit verfügbar sein. Wenn also z.B. eine Bestellung im Online-Shop ausgelöst wird, so muss diese Bestellung nicht unmittelbar in der WaWi vorliegen, sondern es ist hinreichend, dass die Bestellung erst nach einer bestimmten Zeit in die WaWi übernommen wird.
- *Verwendung von Datenobjekten der Integrationskomponenten*:
  Die Integration soll nicht auf der Ebene der Datenhaltung erfolgen (vgl. Abschnitt 2.6.3, S. 36f.), sondern einen möglichst großen Teil der Erzeugung, Validierung und Speicherung der Informationen den zuständigen Datenobjekten der Integrationskomponenten überlassen.
- *Lose Kopplung der Integrationskomponenten*:
  Die zu integrierenden Systeme sollen durch die Integrationslöung in keiner Weise voneinander abhängig werden. Beide Systeme müssen im integrierten Zustand autonom einsetzbar sein.
- *Vollautomatischer Betrieb*:
  Die Integrationslösung soll die Integrationsoperationen vollautomatisch übernehmen. Ein Eingreifen eines Menschen soll nach der Einrichtung nur noch im Fehlerfall erfolgen müssen. Dazu sind auch Funktionen zur Benachrichtigung im Fehlerfall zu berücksichtigen.
- *Minimierung des Betreuungsaufwands für den Online-Shop*:
  Durch den Einsatz der Integrationslösung soll der Aufwand für die Pflege

book

id	isin	titel	autor	preis	regal
1	20340-XX13	Der Sturm	Klaus Haring	12,90	13
2	12340-8B48	Das blaue Kleid	Stella Fair	18,95	13
3	99019-G589	Viel Glück heut	Hans Beinel	21,00	5

Abbildung 3.5: Beispiel eines horizontal modellierten Datenobjekts

des Shops erheblich verringert werden. Der Aufwand zum Einpflegen von Bestellungs- und Kundendaten des Shops in die WaWi soll gänzlich entfallen.

3.5 Datenschnittstellen in Magento Commerce

Die folgenden Abschnitte besprechen die möglichen Schnittstellen, die eine Integrationslösung potentiell nutzen könnte. Sie werden auf ihre Eignung für eine Integration hin untersucht. Die Schnittstellen umfassen die Datenhaltung, die Module DataFlow und Core API, sowie den Magento Appmode.

3.5.1 Magentos Datenbankdesign

Ein technisches Merkmal, das Magento von anderen OpenSource-Shopsystemen unterscheidet, ist das verwendete Datenbankdesign. Viele der Funktionalitäten, die Magento so flexibel und anpassbar machen, aber auch Probleme mit Komplexität oder Performance, haben hierin ihren Ursprung.

Die Abbildung 3.5 zeigt eine herkömmliche Datenbanktabelle, in der der Bücherbestand einer (fiktiven) Buchhandlung gepeichert ist. Alle benötigten Informationen des abzubildenen Objekts Buch sind als eine Tabellenspalte modelliert, in der der entsprechende Wert eingetragen ist. [Kimsal, 2008, 45] nennt das *horizontale Modellierung*. Diese Modellierung ist logisch schlüssig und bildet das Objekt den Anforderungen genügend ab. Allerdings sind *Änderungen der Abbildungsanforderungen*, z.B. das Hinzufügen eines weiteren Attributs, nur möglich, indem die neu benötigten Tabellenspalten ergänzt werden. Bei der horizontalen Modellierung muss also die *Struktur* der Datentabelle in der

book_entity

id	isin
1	20340-XX13
2	12340-8B48
3	99019-G589

book_attribute

id	name	type	display	required
1	titel	varchar	1	true
2	autor	varchar	1	true
3	preis	float	2	false
4	regal	integer	6	false

book_value_varchar

entity_id	attribute_id	value
1	1	Der Sturm
2	1	Das blaue Kleid
3	1	Viel Glück heut
1	2	Klaus Haring
2	2	Stella Fair
3	2	Hans Beinel

book_value_float

entity_id	attribute_id	value
1	3	12,90
2	3	18,95
3	3	21,00

book_value_integer

entity_id	attribute_id	value
1	4	13
2	4	13
3	4	5

Abbildung 3.6: Beispiel eines nach EAV-Model abgebildeten Datenobjekts

Datenbank verändert werden, um bei einer Änderung der Abbildungsanforderungen weitere *Daten* aufnehmen zu können. Ein solcher Eingriff in die Datenbank muss in jedem Fall von Hand durch einen Systemadministrator durchgeführt werden, um Beschädigungen an der Datenbank oder Datenverlust zu vermeiden. Eine flexible, oder gar individuelle Anpassung der Abbildungsanforderungen im Betrieb der Anwendung ist mit der horizontalen Modellierung daher nicht umsetzbar. Durch eine Anpassung der Datenbankstruktur verliert das System u.U. auch seine Updatefähigkeit, da Updates häufig auch die Datenbankstruktur betreffen und in diesem Fall individuell vorgenommene Änderungen überschrieben werden.

Um diese Flexibilität anbieten und nutzen zu können, wurden die Datenbanktabellen für die abzubildenen Objekte in Magento Commerce weitgehend nach dem *Entity-Attribute-Value-Model*, kurz EAV-Model, modelliert. Diese Modellierung, die etwa mit „Objekt-Eigenschaft-Wert-Zuordnung“ übersetzt werden kann, wird auch als *vertikale Modellierung* bezeichnet [Kimsal, 2008, 45]. Attribute eines abzubildenen Objekts werden, nach Datentyp getrennt, als *Datenspalten* in eigenen Tabellen gespeichert. Die Abbildung 3.6 zeigt die Modellierung des Bücherbeispiels aus Abbildung 3.5 als EAV-Model. Das EAV-Model ist vor allem bei Datenhaltungssystemen im medizinischen Bereich sehr verbreitet, wo aus einer sehr großen Anzahl möglicher Attribute für jede Entität eine kleine, aber immer verschiedene Teilmenge gespeichert wird. Mit dieser

Modellierung sind keine Änderungen an der *Struktur* der Datentabellen nötig, wenn durch Änderungen der Anforderungen neue Attribute zu gespeicherten Objekten hinzugefügt werden müssen. Es können jederzeit und unbegrenzt viele selbstdefinierte Attribute ergänzt werden, die bei der Erstellung der Datenbankstruktur noch nicht bekannt waren. Damit können Shopbetreiber in Magento ihre Produktdarstellungen in sehr hohem Maße individualisieren. Es gibt keine vorgegebenen Produkteigenschaften mehr, in die das konkret Angebotene hineingepresst werden müsste. Da jedes Attribut für jedes Produkt einzeln gesetzt wird, kann jedes Produkt eine individuelle Menge von Eigenschaften haben; und schließlich sind auch keine „unprofessionellen" Beschreibungen im Shop mehr nötig, in denen viele Attribute nicht ausgefüllt sind. Trotzdem bleibt das System konsistent und updatefähig, da die Datenbankstruktur unverändert bleibt. Ein anderer Vorteil des EAV-Models ist, dass es die Bereitstellung einer grafischen Benutzeroberfläche zur Verwaltung der Attribute sehr erleichtert [vgl. Kimsal, 2008, 48]. Außerdem ist die Speicherausnutzung sehr effizient. Es werden von jedem Objekt nur die benötigten Attribute gespeichert.

In der Tabelle `book_attribute` in Abbildung 3.6 wurde für die Attribute zusätzlich die Felder `type` und `display` eingeführt. Diese Felder enthalten zusätzliche *Meta-Daten* zu den Attributen, wie deren Datentyp (`type`), die Sichtbarkeit im Shop (`display`) oder ob eine Angabe zwingend erforderlich ist (`required`). Durch Ergänzung weiterer Spalten können weitere nützliche Meta-Daten, bspw. Gruppierungen, Abhängigkeiten, Pflichtangaben usw., abgebildet werden. Diese werden dann von Magento u.a. zur Steuerung der Shopansichten, zur Validierung von Daten und zur Generierung von grafischen Verwaltungsoberflächen verwendet.

Die hohe Flexibilität des EAV-Model hat eine hohe Komplexität der Datenbankstruktur zur Folge. Das physische Schema der Datenmodellierung weicht vom logischen Schema, also der Art in der ein menschlicher Benutzer die Datenstruktur erfasst, stark ab. Für das Beispiel der Modellierung der Bücher werden bei Verwendung des EAV-Models fünf Tabellen benötigt, statt nur einer bei der horizontalen Modellierung, obwohl beide Modellierungen dasselbe logische Schema darstellen. Die Standardinstallation von Magento in der Version 1.3 verwendet dann schon 214 Tabellen. Dem entsprechend viele oder umfangreiche Tabellenabfragen sind nötig, um die modellierten Objekte aus der Datenbank in die Anwendung zu laden. Vor allem beim Auslesen von kompletten Objekten mit allen ihren Attributen müssen viele Tabellen durch JOIN-Operationen verknüpft werden. Bei Tabellen, die sehr viele Zeilen enthalten, oder bei der Suche nach Objekten, deren Attribute bestimmte Werte aufweisen sollen, können die

Abfragen sehr lange brauchen. Die Abfragen, die für solche Zwecke benötigt werden, sind außerdem in der Regel so komplex, dass es für einen Entwickler durchaus schwierig und zeitaufwändig werden kann, eine solche Abfrage selbst zu schreiben. Generell lässt sich für Magento festhalten, dass die Verwendung der Datenbankfunktionalitäten der `Mage_Core` Bibliothek das effektivste Mittel für Datenbankoperationen darstellt [Kimsal, 2008, 48]. Das EAV-Model ist zudem für einen menschlichen Betrachter schwer zu lesen, da der Umfang der gespeicherten Daten nicht aus der Struktur der Tabellen ersichtlich ist. Daraus folgt, dass auch die Erstellung von Schema-Mappings für die Migration bzw. Integration von Daten anderer Systeme sehr komplex ist. Für die Informationsintegration sind Datenbanken nach dem EAV-Model daher nur sehr begrenzt geeignet.

Die aus der Komplexität der Datenbankabfragen begründeten Performanceschwierigkeiten und deren Verringerung bzw. Lösung sind ein wesentliches Optimierungsfeld in Magento Commerce. In der Version 1.3 wurde aus diesem Grund der sog. *Frontend Flat Catalog* eingeführt. Wird dieses Feature aktiviert, so werden für den Produktkatalog und die Produkte des Shops aus den EAV-Tabellen horizontal modellierte Tabellen erzeugt, die alle benötigten Attribute als Spalten enthalten, und die dann ausschließlich für die Anzeige der Produkte und des Katalogs im Shop verwendet werden. In der Release-Meldung spricht Varien von einer möglichen Performancesteigerung von bis zu 40%[7]. Deutliche Steigerungen sind aber vor allem bei umfangreichen Produktkatalogen mit mehr als 1.000 Produkten zu erwarten. Durch den Flat Catalog wird andererseits die Datenhaltung redundant, Änderungen müssen an verschiedenen Orten mehrfach vorgenommen werden, und der Speicherplatzbedarf der Datenhaltung wächst signifikant. Die Einführung des Flat Catalogs kann daher auch als ein Zeichen dafür gewertet werden, dass die Performanceprobleme beim Einsatz von EAV-Datenbanken in Webanwendungen, zu deren wesentlichsten Anforderungen eine schnelle Antwortzeit und ein flüssiges Nutzererlebnis gehört, grundsätzlicher Natur sind und zumindest derzeit und in Verbindung mit der verbreiteten OpenSource-Datenbank MySQL nicht vollständig zufriedenstellend lösbar sind. In der Magento Community sind immer wieder Stimmen zu hören, die eine vollständige Abkehr vom EAV-Model fordern oder vorhersagen[8], von Seiten des Herstellers Varien gibt es dazu zum Zeitpunkt der Fertigstellung dieser Arbeit noch keine Positionierung.

Für die Integrationslösung lässt sich festhalten, dass die Verwendung der Da-

[7] Meldung unter http://www.magentocommerce.com/blog/comments/magento-version-130-is-now-available/, letzter Aufruf am 04.05.2009

[8] http://www.edmondscommerce.co.uk/blog/magento/magento-front-end-flat-catalogue/, letzter Aufruf am 03.05.2009

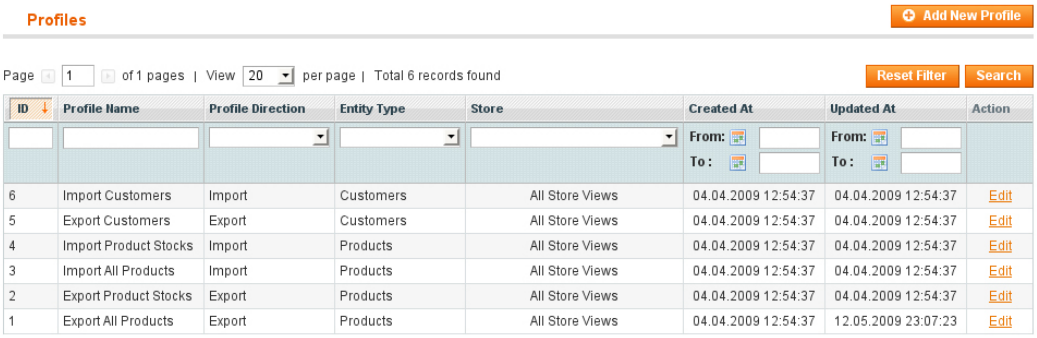

Abbildung 3.7: Liste aller standardmäßig in DataFlow angelegten Profile

tenhaltungsschicht für die Integration mit Magento zwar prinzipiell möglich ist, aber keine empfehlenswerte Lösung darstellt. Das Einfügen von Daten in eine EAV-Datenbank ist noch verhältnismäßig einfach möglich, wenn die komplexe Struktur erst einmal vollständig erschlossen wurde. Hierbei müsste die Integrationslösung allerdings auch umfangreiche Datenvalidierungsfunktionen bereitstellen, um die Daten korrekt einfügen zu können. Das Auslesen von Daten und die Zusammenfassung zu umfassenden Datenobjekten für die Weitergabe an den Integrationspartner würde dann allerdings sehr viel komplexer und umfangreicher sein. Letztlich müssten wesentliche Teile der Ressourcen-Modelle von Magento nachgebaut werden. Eine solche Methode ist weder ökonomisch sinnvoll umsetzbar, noch zukunftssicher, da sich potentiell mit jeder neuen Version von Magento sowohl das Datenbankdesign, als auch die Ressourcen-Modelle ändern können.

3.5.2 Magento DataFlow

Die Datenhaltungsschicht in Magento ist – wie im vorangegangenen Abschnitt ausgeführt wurde – für eine Integration von Daten aus anderen Systemen ungeeignet. Die Fähigkeit zur Integration, bzw. vor allem zur Migration von Daten aus anderen Anwendungen ist jedoch ein essentieller Funktionsbestandteil eines modernen Shops. Gerade auch, wenn Magento Anwender gewinnen möchte, die bereits einen Shop betreiben, muss eine Schnittstelle vorhanden sein, welche die *Migration*, also die einmalige und manuell gesteuerte Übernahme von Daten aus dem anderen System so unkompliziert wie möglich macht. Eine solche Schnittstelle ist in Magento enthalten, sie heißt *DataFlow*. Mit ihr können Produkt- und Kundendaten importiert und exportiert werden.

DataFlow wird vollständig über den Verwaltungsbereich von Magento Commerce im Browser bedient. Es befindet sich im Hauptmenu unter „System

→Import/Export“ und wird über sog. *Profile* gesteuert. Standardmäßig sind bereits eine Reihe von Profilen angelegt, die für verschiedene Import- und Export-Operationen benutzt werden können (siehe Abbildung 3.7). Die Übersicht zeigt zu jedem Profil u.a. die Richtung der Operation sowie den betroffenen Datentyp an. Jedes Profil definiert also eine Operation für ein bestimmtes Datenobjekt (Kunden- oder Produktdaten) in einer bestimmten Richtung (Import oder Export). Über eine Reihe von Eingabefeldern können neben dem Datenobjekt und der Operationsrichtung auch das Format und die Herkunft, bzw. der Ablageort der Datei mit den Daten angegeben werden. Bei einem Exportprofil können zusätzlich noch verschiedene Filter verwendet werden, um z.B. nur einen gewünschten Teil des Gesamtdatenbestands zu exportieren (siehe Abbildung 3.8). Als Dateiformate für den Datenaustausch werden von DataFlow das Format →CSV und ein spezielles Excel XML-Format unterstützt. Um mit DataFlow eine Operation auszuführen, muss das entsprechende Profil aufgerufen und aktiviert werden. Im Zuge der Ausführung müssen häufig von Hand Einstellungen, wie die Angabe des Ablageortes einer Datei o.ä., getätigt werden.

Mit DataFlow verfügt Magento Commerce über eine komfortable Schnittstelle, um Produkt- und Kundendaten zu exportieren und zu importieren. Sie ist aber eher für Migrationszewcke bestimmt, d.h. für Fälle, in denen der Bestand eines anderen Shopsystems an Kunden und Produkten *einamlig* und *manuell initiiert* komfortabel eingefügt werden soll. Ein Beispiel für eine solche Datenmigration aus dem Shopsystem xt:Commerce wird in [Zenner, 2009, 319ff] anschaulich beschrieben. Für das konkrete Integrationsvorhaben im Rahmen dieser Arbeit jedoch ist DataFlow aus mehreren Gründen nicht geeignet. Zum Einen fehlt die Möglichkeit mit den Bestellungs- und Katalogdaten zu arbeiten. Beide Datenobjekte werden von DataFlow derzeit nicht unterstützt. Zum Anderen ist DataFlow nicht automatisch ansprechbar. Alle Operationen erfordern zuerst, dass ein Benutzer sich im Verwaltungsbereich anmeldet und anschließend muss dieser verschiedene Schaltflächen klicken und weitere Angaben machen, ohne denen die Operation nicht durchgeführt werden kann. Eine andere, nicht grafisch gebundene Interaktionsmöglichkeit mit DataFlow ist nicht vorhanden, eine automatisierte Nutzung dieser Schnittstelle ist damit für die Integrationslösung im Rahmen dieser Arbeit nicht sinnvoll möglich.

3.5.3 Magento Core API

Magento Commerce bietet mit der *Core API* seit der Version 1.1 eine umfangreiche →API zur Verwendung vieler Anwendungsfunktionalitäten von au-

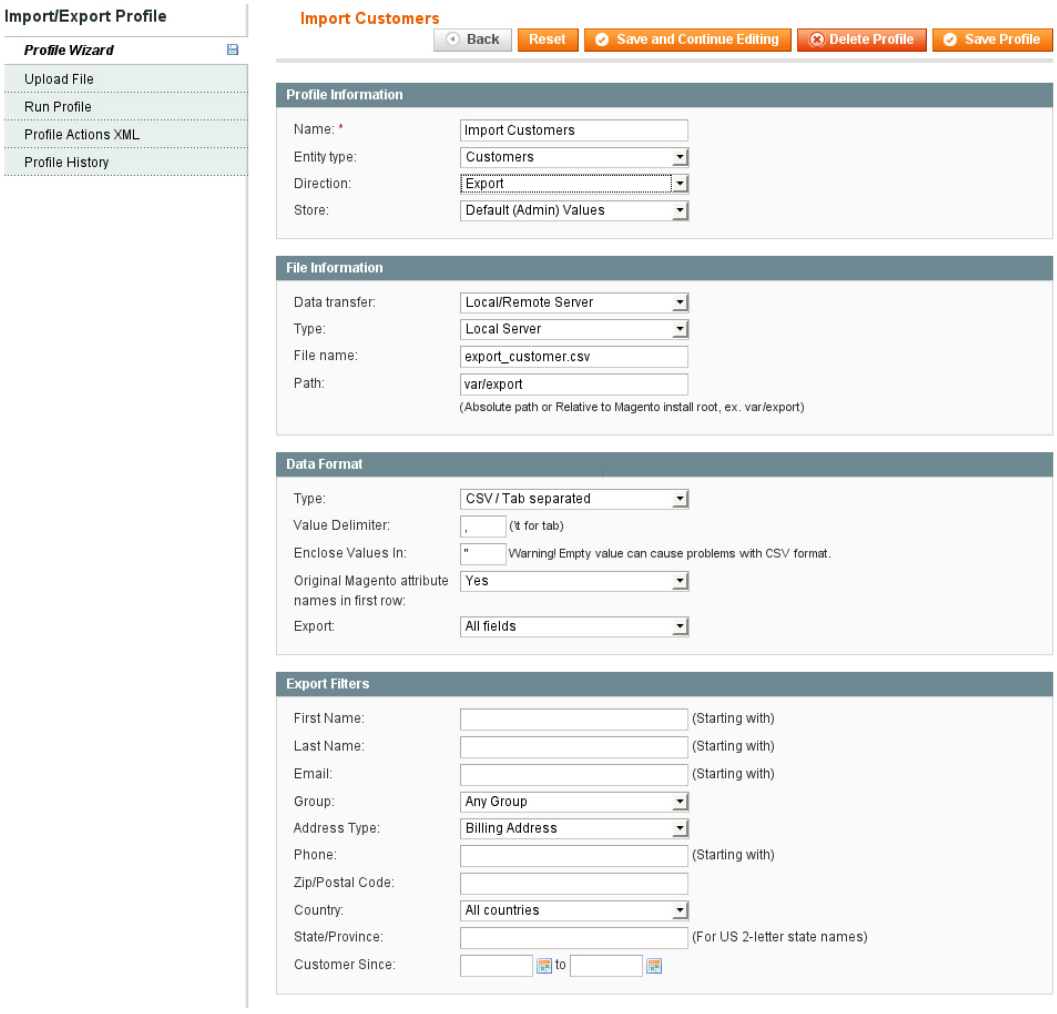

Abbildung 3.8: DataFlow-Eingabemaske mit Filtern für eine Exportoperation

```
1  // SOAP-Client starten
2  $client = new SoapClient('http://magentodomain/api/soap/?wsdl');
3
4  // als API-Benutzer bei Magento anmelden
5  $session = $client->login('apiUser', 'apiKey');
6
7  // Bestellungen abrufen
8  $result = $client->call($session, 'sales_orders.list',
9      array(array(
10         'created_at' => array('from' => '2009-01-01 12:00:00')
11     ))
12 );
13
14 // Bestellungen anzeigen
15 print_r $result;
16
17 // abmelden, wenn keine weiteren Aufrufe mehr
18 $client->endSession($session);
```

Tabelle 3.1: Anwendungsbeispiel für die Magento Core API: Abruf aller Bestellungen seit einem gegebenen Datum

ßerhalb der Anwendung an. Die API ist als Web Service (vgl. Abschnitt 2.7.1, S. 40) ausgeführt, kann sowohl mit dem →SOAP als auch mit dem →XMLRPC Format arbeiten und ist daher von anderen Anwendungen, unabhängig von deren Plattform und Programmiersprache, leicht zu benutzen. Die API erlaubt den Zugriff auf Daten der Magentomodule *Customer* (Kundendaten), *Catalog* (Produkt-, Katalog- und Inventardaten) und *Order* (Bestellungs- und Bezahlungsdaten)[9]. Es können jeweils sowohl die existierenden Daten ausgelesen, als auch neue Datensätze angelegt werden. In Magento können für die API verschiedene Nutzungsprofile angelegt werden, die mit unterschiedlichen Rechten ausgestattet sein können. Damit ist es z.B. möglich, dass ein API-Benutzer nur Bestelldaten aus Magento exportieren darf, oder dass ein anderer Benutzer ausschließlich Katalogdaten nach Magento importieren kann.

Der große Vorteil einer API liegt in der einfachen Benutzung. Sämtliche nötigen Prozesse zur Validierung der Daten, zur Speicherung in der Datenhaltungsschicht, sowie zum Auslesen aus der Datenhaltungsschicht laufen für den API-Benutzer unsichtbar im Hintergrund der Anwendung ab. Dabei werden größtenteils dieselben Programmkomponenten verwendet, die auch die Prozesse bei der direkten Interaktion mit dem Frontend oder Backend der Anwendung steuern. Vereinfacht gesagt simuliert also ein API-Aufruf eine Nutzeraktion im Verwaltungsbereich der Anwendung.

[9] Dokumentation der Core API, sowie eine Auflistung aller erlaubten Aufrufe und Parameter unter http://www.magentocommerce.com/support/magento_core_api, letzter Aufruf am 11.05.2009

Anders als etwa beim Bearbeiten von Daten in der Datenhaltungsschicht werden bei Verwendung der API alle von der Anwendung definierten Anforderungen und Abläufe angewendet. Komplexe Abläufe wie bspw. das Speichern und Abrufen von Informationen in der EAV-Datenbank geschehen für den API-Benutzer transparent, d.h. unsichtbar im Hintergrund. Das nimmt dem API-Benutzer Arbeit ab, der für einen API-Aufruf nötige Programmcode wird sehr kurz und übersichtlich, und schließlich stellt jeder API-Aufruf ein definiertes Resultat (Fehler- oder Statuscode, sowie ggf. angeforderte Daten) bereit. Für den Anbieter der API ist gleichzeitig gesichert, dass keine unerlaubten Datenmanipulationen vorgenommen werden können.

Im Hintergrund verwendet Magento das PHP-Modul `SoapServer` um die API-Funktionalitäten zur Verfügung zu stellen. Es empfängt eingehende SOAP-Anfragen von Clients, ruft die in der Anfrage geforderte Funktion mit den ebenfalls aus der Anfrage erhaltenen Parametern auf. Diese Funktion ermittelt die zuständige Magentoklasse, ruft sie auf und erhält als Rückgabe ein Array mit Daten. Diese Ergebnisdaten werden anschließend in einer SOAP-Meldung verpackt an den Client zurückgeschickt.

Die Core API ist das für die Integration von Magento am besten geeignete Werkzeug. Sie ist speziell auf den Export bzw. Import der entsprechenden Daten ausgerichtet, ermöglichst abgestufte Berechtigungen für verschiedene API-Benutzer und ist von außerhalb der lokalen Anwendungsumgebung zu erreichen. Zwei Schwachpunkte lassen sich dennoch ausmachen: Zum einen ist die API relativ langsam. Da der Web Service das HTTP-Protokoll als Transportprotokoll verwendet, und zudem jeder API-Aufruf einen kompletten Anwendungszyklus in Magento auslöst, können Integrationsaktionen, die eine große Anzahl von API-Aufrufen auslösen, schnell eine sehr lange Ausführungszeit benötigen. Das Entwicklerteam hat jedoch angekündigt, dass die Verbesserung der Performance der API ein wesentlicher Optimierungspunkt auf ihrer Agenda ist, und voraussichtlich noch im Jahr 2009 signifikante Fortschritte erzielt werden [Zenner, 2009, 330f] können. Das zweite Problem ist allerdings grundlegender: das HTTP-Protokoll überträgt alle Informationen unverschlüsselt. Das bedeutet, dass alle übertragenen Daten und auch die verwendeten API-Zugangsdaten im Klartext mitlesbar sind. Zu den Daten gehören auch hochsensible Informationen, wie Kundenadressen oder Konten- und Kreditkartendaten. Es ist daher für den Einsatz in Produktivumgebungen mit echten Daten unbedingt nötig, ein sicheres Protokoll wie z.B. HTTPS für die Übertragung der API-Aufrufe zu verwenden. Dazu muss der Server, auf dem Magento betrieben wird, entsprechend eingerichtet sein.

```
1  // Magento Hauptdatei einbinden
2  $magentoPath = '/var/www/magento/'
3  require_once($magentoPath.'app/Mage.php');
4
5  // Magento Appmode laden
6  Mage::app('default');
7
8  // API Model für Bestellungen laden
9  $orders = Mage::getModel('sales/order_api');
10
11 // Bestellungen abrufen
12 $result = $orders->items(array(
13     'created_at' => array('from' => '2009-01-01 12:00:00')
14 ));
15
16 // Bestellungen anzeigen
17 print_r $result;
```

Tabelle 3.2: Anwendungsbeispiel für den Magento Appmode: Abruf aller Bestellungen seit einem gegebenen Datum

3.5.4 Magento Appmode

Magento bietet noch einen weitere Möglichkeit der Integration über die Anwendungsschicht. Magento kann im sog. *Appmode* aufgerufen werden. Anders als bei einem normalen Aufruf von Magento wird dann nicht der komplette Anwendungszyklus durchlaufen und als Resultat z.B. ein HTML-Dokument ausgegeben (sog. Runmode). Statt dessen wird die komplette Anwendung mit allen Komponenten lediglich initialisiert und in einem Objekt gespeichert, das dann für weitere Operationen zur Verfügung steht. Innerhalb des Objekts können dann dieselben Klassen und Komponenten, die bei einem Aufruf über die Core API verwendet würden, direkt angesprochen und verwendet werden, unter Umgehung des HTTP-Requests und des SOAP-, bzw. XML RPC Formats. Damit kann dieselbe Funktionalität verwendet werden, wie bei der Core API, jedoch viel schneller, da der zeitraubende Teil des Versendens der WebService-Requests entfällt.

Das Appmode-Verfahren hat als Verfahren der Anwendungsintegration prinzipiell die gleichen Vorteile wie die Verwendung der Core API. Alle von der Anwendung definierten Anforderungen und Abläufe werden verwendet, die Daten werden optimal aufbereitet und validiert, die Speicherung bzw. der Abruf von Informationen in der EAV-Datenbank ist für den Integrationsprozess transparent. Der wesentliche zusätzliche Vorteil des Appmode-Verfahrens ist seine Geschwindigkeit. Durch den Wegfall des HTTP-Requests und die Umgehung der für das Erstellen und Parsen der SOAP-Nachrichten nötigen Komponenten ist das Verfahren deutlich schneller als die Verwendung der Core API. Nicht

repräsentative Tests während der Entwicklung der Integrationslösung zeigten einen Geschwindigkeitsvorteil von mindestens 60% gegenüber der Core API. Es kann außerdem angenommen werden, dass der Geschwindigkeitsvorteil bei Operationen mit vielen Teilabfragen noch deutlich höher ausfallen könnte.

Allerdings hat das Appmode-Verfahren auch Einschränkungen gegenüber der Core API. Das Appmode-Verfahren ist nur dann anwendbar, wenn sich die Integrationslösung und das Magento-System im selben Adressraum auf derselben lokalen Umgebung, d.h. konkret in derselben PHP-Instanz, befinden. Anderenfalls kann die Integrationslösung nicht auf die im Speicher initialisierten Magento-Objekte zugreifen. Außerdem hat die Integrationslösung bei Verwendung des Appmode-Verfahrens deutlich mehr Handlungsoptionen. Theoretisch stehen ihr im Appmode alle Methoden zur Verfügung, die von Magento-Klassen als öffentlich definiert wurden. Im Gegensatz zur Core API gibt es auch keine Berechtigungssteuerung. Wenn der Appmode einmal gestartet ist, stehen der Integrationslösung sozusagen „alle Möglichkeiten offen“.

3.6 Datenschnittstelle in ALEA Commerce Suite

Als Schnittstelle für den Export von Daten aus der Warenwirtschaft, bzw. für den Import in die Warenwirtschaft dienen XML-Austauschdateien. Diese werden jeweils auf dem System der Datenquelle erzeugt und über eine Sycnronisationsschnittstelle (`rsync`, vgl. Abschnitt 4.2, S. 104) auf das andere System transferiert. Dort werden die Daten wieder aus der XML-Datei herausgelesen und anschließend im System verarbeitet. Die XML-Dateien sind in UTF-8 kodiert, um alle gängigen Sonderzeichen bei der Übertragung berücksichtigen zu können. Da UTF-8 das Encodingformat ist, in dem Magento Commerce Daten speichert und verarbeitet, müssen die Daten in Mageport nicht in eine andere Zeichenkodierung umgewandelt werden. Für das Root-Tag kann die Adresse einer XML-Schemadatei angegeben werden, gegen die die XML-Datei dann vor der Auswertung durch ALEA Commerce Suite validiert werden kann. Damit kann sichergestellt werden, dass die Struktur der Datei korrekt ist, und nur erlaubte Datenattribute enthalten sind. Weitere Maßnahmen zur Sicherung des Inhalts z.B. durch CDATA-Kommentare[10] sind nicht vorgesehen, da keine

[10] Enthält ein XML-Tag einen String, der selbst Zeichen enthält, die in XML verwendet werden (hauptsächlich die Zeichen < und >), so wird dieser String mit einem CDATA Kommentar der Form `<![CDATA[ Inhalt ]]>` umschlossen. Somit werden alle Zeichen als Teil des Strings interpretiert. Die Abkürzung CDATA steht für Character Data, bedeutet

Attributwerte mit solchen formalen Besonderheiten zu erwarten sind.

Die für die Integration relevanten Datenobjekte und deren Attribute werden innerhalb der XML-Datei so bezeichnet und strukturiert, wie es für ALEA Commerce Suite geeignet ist, sie mühelos weiter zu verarbeiten. Da die Integrationslösung für das Auffinden von Datenattributen in den XML-Austauschdateien eine eigene Mappingdatei verwendet, sind alle Bezeichnungen von Tags und deren Verschachtelungsstruktur von der Warenwirtschaft völlig frei bestimmbar. Der Integrationslösung muss lediglich eine strukturell identische Mappingdatei vorliegen. Weitere Ausführungen zum Aufbau und zur Erstellung der Mappingdatei befinden sich in Abschnitt 4.6, S. 117.

3.7 Analyse der Datenschemata

3.7.1 Datenschema für Magento Commerce

Zunächst könnte man erwarten, dass sich die Datenschemata in den beiden zur Verfügung stehenden Schnittstellen in Magento Commerce voneinander unterscheiden. Wenn die blanke Rückgabe des jeweiligen Schnittstellenaufrufs betrachtet wird, trifft das auch zu: Bei Verwendung der Core API wird ein SOAP-Dokument zurückgegeben, also ein nach speziellen Regeln aufgebautes XML-Dokument, in dessen Inhaltsbereich die Daten in Form eines definierten SOAP-Datentyps (Soap ComplexType) geschrieben sind. Dieses SOAP-Dokument und die darin enthaltenen Daten werden jedoch dem Entwickler nicht in roher Form zur Verfügung gestellt. Vielmehr erfolgt bei der Verwendung des PHP-Moduls `SoapClient`, dem Standardwerkzeug für die Verwendung von SOAP-APIs in PHP, sofort eine Umwandlung der enthaltenen Daten in ein – gemäß der Hierarchie und Struktur der Daten verschachteltes – Array. Dieses Array gibt `SoapClient` zurück und damit kann dann weitergearbeitet werden. Bei Verwendung des Magento Appmode dagegen werden alle Schritte, die der Erzeugung, Übertragung und Umwandlung der SOAP-Nachrichten dienen, einfach ausgelassen. Die Rückgabe ist stattdessen ebenfalls ein verschachteltes Array. Dieses Array hat exakt die gleiche Struktur und den gleichen Aufbau wie das Array, das von `SoapClient` ausgegeben wird. Das sollte auch so sein, denn schließlich verwendet der Appmode die gleichen API-Klassen, die auch die Core API benutzt. Die Core API verwendet außerdem zum Erstellen und Senden der SOAP-Dokumente selbst den `SoapServer`, der das Gegenstück zum `SoapClient` Modul darstellt. Das vom `SoapClient` gelieferte Array

also „Zeichendaten“.

ist dasselbe, das dem `SoapServer` in Magento zum Verschicken übergeben wurde. Daher müssen sich die Schemata der Datenrückgaben beider Schnittstellen gleichen.

```
array(
   [0] => array(
      'id'       => 123,
      'name'     => 'Max Mustermann',
      'adresse' => array(
          'privat1' => array(
             'straße' => 'Musterstraße 12',
             'ort'    => 'Musterstadt'
          ),
          'privat2' => array(
             'straße' => 'Dorfgasse 8',
             'ort'    => 'Musterdorf'
          )
      )
   ),
   [1] => array(
      'id'       => 124,
      'name'     => 'Frauke Schneider',
      'adresse' => array(
          'privat' => array(
             'straße' => 'Wendelweg 1',
             'ort'    => 'Musterstadt'
          )
      )
   ),
 [...]
)
```

Tabelle 3.3: Beispiel für ein Magento-Ergebnisarray

Für die Integrationslösung hat dies die Auswirkung, dass sich die Wahl der Schnittstelle zu Magento in keiner Weise auf die Daten auswirkt, die zurückgeliefert werden. Die Integrationslösung kann somit auf ein einheitliches Datenschema aufbauen, das von beiden Schnittstellen bedient wird. Es werden keine Umwandlungs- oder Angleichungsoperationen nötig, die Integrationslösung spart an Umfang und Komplexität.

Das Datenschema für die Daten von Magento Commerce sieht also wie folgt aus: Für jeden Grunddatentyp (Kundendaten, Bestellungsdaten, Produkt- und Katalogdaten) gibt es jeweils ein eigenes Array. Dieses Array ist das Gesamtergebnis einer an die Schnittstelle gerichteten Anfrage. Es entspricht dem logischen Konzept „eine Menge von ...“, stellt also einfach einen Container dar. Jedes Feld dieses Container-Arrays ist wiederum von einem Array belegt. Dieses Array stellt jeweils ein konkretes Datenobjekt dar, also z.B. einen Kunden. Innerhalb dieses Arrays sind nun die konkreten Attribute und Eigenschaften des einzelnen Datenobjekts definiert. Jedes Feld hat einen eindeutigen Namen, der das Attribut benennt, und einen zugewiesenen Wert, der den konkreten Wert des Attributs darstellt. Je nach Komplexität des Datenobjekts ist

dieses Array in sich weiter verschachtelt. Die Abbildung 3.3 zeigt ein fiktives Beispielarray für die Abfrage nach Kunden, das die beschriebene Art der Verschachtelung aufweist. Die konkrete Gestalt der einzelnen Arrays soll hier nicht dargestellt werden. Für einen ungefähren Eindruck können aber die Tabellen im Anhang A.1, Seite XVI dienen. Diese zeigen alle von der Integrationslösung verwendeten Datenattribute und in der dritten Spalte deren Entsprechungen in den Magento-Ergebnisarrays, und deuten deren Verschachtelung durch kleine Pfeile an. Die tatsächlich von Magento zurück gegebenen Arrays enthalten noch eine Reihe weitere Attribute, die für die Integrationslösung im Rahmen dieser Arbeit entweder nicht benötigt oder nicht berücksichtigt wurden.

3.7.2 Datenschema für ALEA Commerce Suite

Die Daten, die für das System der Warenwirtschaft verarbeitbar sein sollen, liegen im XML-Format in Form einer verschachtelten Baumstruktur vor. Die Verschachtelung der Datenobjekte und Attribute ist der Verschachtelung der Datenattribute in den Arrays von Magento Commerce prinzipiell sehr ähnlich, da sie dieselben Eigenschaften repräsentieren. Grundsätzlich wird jedes Datenattribut durch ein Tag repräsentiert, der Wert des Tags ist die konkrete Ausprägung des Attributs für das einzelne Objekt. Es gibt jedoch einige Besonderheiten in der Struktur der XML-Datei, die von der Struktur der Arrays abweichen. Die Tabelle 3.4 zeigt ein fiktives und unvollständiges XML-Dokument, anhand dessen die Besonderheiten jedoch gut zu erkennen sind.

Die erste Besonderheit liegt in den zusätzlich zu den eigentlichen Daten vorhandenen Elementgruppen und Wurzeltags. Da die Austauschdateien von der Warenwirtschaft oft auch zu anderen Zwecken verwendet werden, z.b. um eine andere unternehmensinterne Anwendung anzusprechen, sind in diesen Dateien häufig weitere Vorgaben und Elemente definiert um diese Funktionen nutzen zu können. Im Beispiel sind dies die Angaben des Tags `<metaData>`. Diese Angaben muss die Integrationslösung also ignorieren, sie dürfen aber nicht zu einem Fehler führen. Dazu kommt, dass die Wurzelelemente `<wawiInterchangeDocument>` und `<salesOnline>`, sowie auch das Container-Element `<data>`, nicht zwingend immer vorhanden sind. Eine andere Warenwirtschaft könnte ganz andere Vorgaben hinsichtlich der Anordnung der Daten innerhalb ihrer Austauschdateien machen. Die Integrationslösung verwendet daher nicht die Struktur der XML-Datei an sich für das Auslesen der Daten. Stattdessen definiert sie Schlüssel-Tags, die sog. *Group Tags*, mit denen nach bestimmten Container-Elementen im XML-Dokument gesucht wird.

```
<?xml version="1.0" encoding="utf-8"?>
<wawiExchangeDocument>
   <salesOnline>
      <metaData>
         <created>2009-04-31 12:19</created>
         <purpose>Import to company ERP</purpose>
      </metaData>
      <data>
         <orders>
            <order>
               <created>2009-04-22 18:25</created>
               <currency>EUR</currency>
               <total>122.87</total>
               <customer>
                  <id>123</id>
                  <email>customer@domain.de</email>
               </customer>
               <articles>
                  <item>
                     <id>8291</id>
                     <quantity>2</quantity>
                     <singlePrice>12.90</singlePrice>
                     <itemTotal>25.80</itemTotal>
                  </item>
                  <item>
                     <id>129</id>
                     <quantity>1</quantity>
                     <singlePrice>69.89</singlePrice>
                     <itemTotal>69.89</itemTotal>
                  </item>
               </articles>
            </order>
            <order>
 [...]
            </order>
         </orders>
      </data>
   </salesOnline>
</wawiExchangeDocument>
```

Tabelle 3.4: Strukturbeispiel für eine XML-Austauschdatei mit Bestellungsdaten

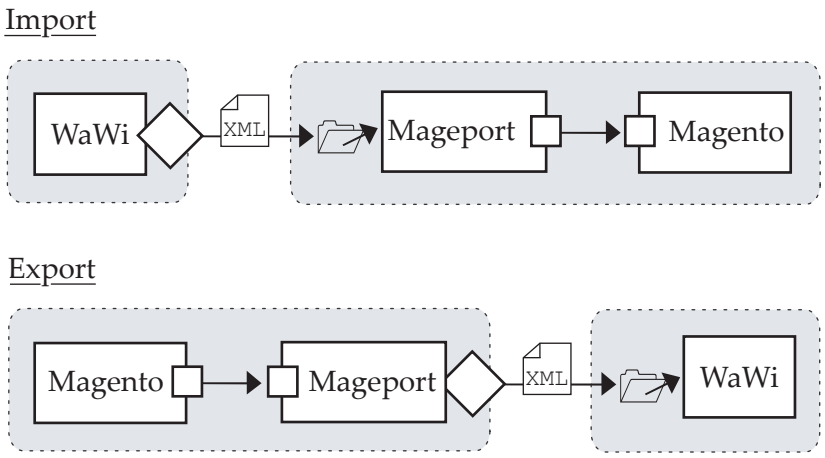

Abbildung 3.9: schematischer Aufbau der Integrationslösung

Diese werden dann als Träger der relevanten Daten angenommen werden (siehe Abschnitt 4.6, S. 117). In der Tabelle 3.4 ist das Tag `<orders>` ein solches Group Tag. Nur die Daten innerhalb dieser Container werden verwendet.

Innerhalb eines Containers sind verschiedene Arten von Gruppierungen zu erkennen. Datenelemente, die selbst wieder ein Objekt darstellen, wie z.B. ein Produkt oder eine Adresse, werden in einem XML-Tag abgebildet, dass als Kindelemente die Attribute dieses Objekts enthält. In der Tabelle 3.4 ist dies z.B. das Objekt `<order>`, oder innerhalb dessen das Unterobjekt `<customer>`. Andere Tags enthalten als Kindelemente eine Reihe von gleichartigen Objekten und dienen als Container für diese. Neben dem Group Tag ist dies im Beispiel das Tag `<articles>`. Diese Container dürfen nur gleichartige Unterobjekte enthalten.

3.8 Abgeleitete Integrationslösung

3.8.1 Prinzipieller Aufbau

Die Abbildung 3.9 zeigt den schematischen Aufbau der Integrationslösung und die prinzipiellen Abläufe bei der Integration in beide Richtungen. Die Integrationskomponenten Magento Commerce und ALEA Commerce Suite sind voneinander räumlich und zeitlich komplett autark. Dies ist daraus begründet, dass die Anwendungen auf verschiedenen physischen Systemen betrieben werden. Die Warenwirtschaft als betriebsinterne Anwendung wird außerdem vom

öffentlich zugänglichen Internet deutlich strenger isoliert sein, als der Online-Shop, der über das Internet erreichbar sein muss. Die Integrationslösung fungiert als vermittelnde Instanz. Sie kann sowohl autark von beiden Komponenten, als auch auf demselben lokalen System wie der Online-Shop Magento Commerce betrieben werden. Sie übernimmt die Umsetzung der Datenformate in das vom jeweiligen Empfänger benötigte Format, sowie die Übergabe der Daten. Sowohl das Format, als auch die Übergabe der Daten unterscheiden sich bei den Integrationskomponenten. Die Warenwirtschaft ALEA Commerce Suite liefert und erwartet Daten als XML-Dateien, die auf die Umgebung der Integrationslösung hochgeladen, bzw. von dort abgerufen werden. Die Kommunikation mit dem Online-Shop Magento Commerce funktioniert prinzipiell über zwei verschiedene Kanäle: In jedem Fall kann die externe *Core API* benutzt werden. Damit kann auch ein Magento-System verwendet werden, das sich nicht in einer lokalen Umgebung mit der Integrationslösung befindet. Sind beide jedoch auf einem lokalen System, so kann Magento direkt, d.h. über den sog. *Appmode*, angesprochen werden (vgl. Abschnitt 3.5.4, S. 87). Dieses Kommunikationsverfahren bietet einen großen Geschwindigkeitsvorteil gegenüber einer externen Verwendung der Core API.

Eine der grundlegenden Entscheidungsfragen für den Aufbau der Integrationslösung ist die der Autonomie. Grundsätzlich ist es sowohl denkbar, die Integrationslösung fest in Magento Commerce zu integrieren, bspw. als Modulerweiterung, als auch, sie als eigenständiges Skript von Magento unabhängig auszuführen. Die Ausführung als Modulerweiterung hat Vorteile vor allem bei der Verwendung von vorhandenen Funktionen in Magento, bzw. im darunter liegenden → Zend Framework. Viele grundlegende Funktionen sind dort bereits vorhanden und können bequem wiederverwendet werden. Das umfasst z.B. Funktionen zum Parsen und Erstellen von XML-Dateien, oder zum Versenden von Benachrichtigungen. Außerdem ist ein Modul in Magento leicht an alle anderen Funktionen anbindbar, und kann so bspw. in Reaktion auf sog. *Events* nach bestimmten Nutzeraktionen im Shop von selbst aktiv werden. So kann bei diesem Lösungsansatz etwa eine Bestellung direkt nach der Erzeugung an den Integrationspartner weitergeleitet werden.

Der Lösungsansatz eines unabhängigen Skriptes nutzt hingegen vor allem die Vorteile, die sich aus der Autonomie ergeben. Das unabhängige Skript kann z.B. in einem von Magento unabhängigen Prozess ablaufen, und damit wird die Performance des Shops nicht durch zeitgleich ablaufende Integrationsoperationen belastet. Das unabhängige Skript kann problemlos von außen aufgerufen werden, ohne dass weitere Aktivitäten in Magento ausgelöst werden. Das verschlankt die Abläufe und verkürzt die benötigte Ausführungszeit. Deswei-

teren kann das Skript in diesem Fall so gestaltet werden, dass sich prinzipiell auch andere Onlineshop-Systeme damit ansprechen ließen. Weiterhin hat ein Großteil der Arbeitsschritte der Integrationslösung nichts mit den spezifischen Abläufen in Magento zu tun. Eine Trennung der beiden ist aus der Sicht der Anwendungslogik daher sinnvoll. Aus diesen Gründen wurde für die Integrationslösung im Rahmen dieser Arbeit der Ansatz über ein eigenständiges Skript verwirklicht.

Das Skript wird für die Ausführung auf der Kommandozeile mit dem → CLI-Binary von PHP ausgelegt. Dadurch entfallen verschiedene Beschränkungen, denen das Skript bei einem Aufruf im Browser unterworfen ist [vgl Sklar / Trachtenberg, 2005, 660]. U.a. ist die maximale Ausführungszeit des Skripts im CLI-Binary nicht beschränkt. Dies ist besonders bei umfangreichen Operationen hilfreich, die – vor allem auch bei Verwendung der Magento Core API – u.U. eine sehr lange Zeit benötigen. Das Skript muss auch nicht über das Internet erreichbar sein. Stattdessen kann es einfach von einem sog. *Cronjob*, in regelmäßigen Abständen aufgerufen und ausgeführt werden. So ist ein vollautomatischer Betrieb komfortabel umsetzbar, und es ist sichergestellt, dass kein Unbefugter das Skript von außerhalb ansprechen kann. Für Entwicklungs- und Testzwecke unterstützt das Skript aber auch den Aufruf über den Browser und die Verarbeitung duch das → CGI-Binary von PHP. Hier können insbesondere Fehlermeldungen und Debug-Ausgaben wesentlich komfortabler gelesen werden, als auf der Kommandozeile.

3.8.2 Umsetzung in PHP

Die Integrationslösung wird in der Skriptsprache → PHP umgesetzt. Für diese Entscheidung gibt es eine Reihe von Gründen, die in diesem Abschnitt dargelegt werden. Zum ersten ist die Verwendung von PHP nötig, um den *Magento Appmode* nutzen zu können (siehe Abschnitt 3.5.4, S. 87). Bei der Verwendung des Appmode werden die benötigten Objekte von Magento direkt im Adressraum des Skriptes erzeugt. Da Magento in PHP geschrieben ist, muss auch die Integrationslösung PHP verwenden. Daneben ist PHP aber auch ein für den Anwendungszweck der Integrationslösung geeignetes Werkzeug. PHP arbeitet hervorragend mit dem Apache-Server und Linux-Betriebssystemen zusammen, die auch Magento verwendet. Durch seine mächtige Standardbibliothek bietet es optimale Methoden für viele Operationen bereits integriert an. Im Zusammenspiel mit zusätzlichen PEAR-Bibliotheken kann für sämtliche grundlegenden Aufgaben auf bewährte und rubuste Algorithmen zurückgegriffen werden.

Das beschleunigt und erleichtert die Umsetzung sehr. [vgl Rechenberg / Pomberger, 2006, 1124f] Mit dem → CLI-Binary, das zum Standardumfang von PHP zählt, existiert zudem eine komfortable Möglichkeit, um die Integrationslösung unabhängig vom Browser und isoliert vom öffentlich zugänglichen Internet zu betreiben. Durch die Objektorientierung seit der Version 5 ist die entstehende Integrationslösung modular aufbaubar, flexibel erweiterbar und relativ leicht zu warten. Auf die Grundlagen von PHP kann im Rahmen dieser Arbeit nicht eingegangen werden. Für eine weitere Beschäftigung sei deshalb auf Sklar / Trachtenberg [2005] und Welling / Thomson [2009], sowie auf die offizielle Website des PHP-Projekts[11] verwiesen.

3.8.3 Funktionsumfang

Die Kernfunktion der Integrationslösung liegt in der Aufbereitung und dem Austausch von Daten zwischen den Integrationspartnern ALEA Commerce Suite und Magento Commerce. Dabei gibt es zwei Hauptrichtungen, in die die Daten fließen können (siehe auch Abbildung 3.9, Seite 93). Im folgenden soll von *Export* gesprochen werden, wenn Daten aus Magento Commerce herausgelesen und für ALEA Commerce Suite aufbereitet werden. Datenoperationen in die entgegengesetzte Richtung sollen als *Import* bezeichnet werden. In der Benennung spiegelt sich auch die Tatsache wieder, dass die Integrationslösung wesentlich enger mit Magento verknüpft ist, als mit der Warenwirtschaft. Export und Import beziehen sich auf die jeweilige Operation, die im Zuge der Integrationsfunktion in Magento vollzogen wird.

Export und Import sind die beiden umfassenden Funktionen der Integrationslösung. Sie sind nicht gleichzeitig durchführbar und unterscheiden sich grundlegend in Ablauf und Ergebnis. Sie stellen damit die beiden Befehle dar, die das Skript akzeptiert. Jedem der beiden Befehle kann über angehängte Parameter mitgeteilt werden, welche Datenobjekte der Befehl berücksichtigen soll. Eine Auflistung der berücksichtigten Parameter und der dahinter stehenden Datenobjekte findet sich in den folgenden beiden Abschnitten. Mit der Angabe des Parameters `-help` können außerdem alle verfügbaren Parameter zu einem der Befehle angezeigt werden.

[11] http://php.net/, letzter Aufruf am 16.05.2009

3.8.3.1 Exportfunktionen

Die Exportfunktionen akzeptieren als zusätzlichen Parameter die Angabe von `-since`. Dieser Parameter übergibt einen Zeitpunkt, ab dem die Operationen greifen sollen. Es werden dann nur Datensätze herangezogen, die sich seit dem gegebenen Zeitpunkt verändert haben. Somit werden bei Bedarf nur aktuelle Änderungen der Datenobjekte berücksichtigt. Neben dem Parameter `-since` gibt es noch eine weitere Möglichkeit, nur aktuelle Daten berücksichtigen: Wenn in der Konfigurationsdatei die Einstellung `<saveExecutionTime>` aktiviert ist, wird bei jedem Aufruf einer Operation die aktuelle Zeit gespeichert. Ist `<useExecutionTime>` aktiviert, wird diese gespeicherte Zeit beim nächsten Aufruf als Parameter von `-since` verwendet, wenn dieser nicht angegeben ist.

Export von Kundendaten Der Export von Kundendaten wird durch die Angabe des Parameters `-customers` (Kurzschreibweise `-c`) an den Befehl `Export` angestoßen. Ist zusätzlich der Parameter `-since` gesetzt, werden alle Kunden, deren Daten sich nach dem angegebenen Zeitpunkt verändert haben, exportiert. Fehlt der Parameter, so werden alle in der Datenbank vorhandenen Kunden exportiert. Die Daten, die zu jedem Kunden exportiert werden, umfassen Stammdaten, wie Kunden-ID, Name oder Emailadresse, sowie eine beliebige Anzahl an Adressen mit jeweils eigenen Namens- und Kontaktattributen. Die einzelnen Attribute, die zu jedem Kunden erfasst werden, können der Tabelle A.2 im Anhang A.1, Seite XVII entnommen werden.

Export von Bestellungsdaten Daten über im Online-Shop gespeicherte Bestellungen werden mit dem Befehl `Export` mit angehängtem Parameter `-orders` (Kurzschreibweise `-o`) verarbeitet. Die Angabe des Parameter `-since` bewirkt hier, dass nur Bestellungen berücksichtigt werden, die nach dem angegebenen Zeitpunkt getätigt wurden. Ist der Parameter nicht vorhanden, werden alle gespeicherten Bestellungen berücksichtigt. Jede Bestellung umfasst eine Reihe von Datenobjekten und Datenattributen. Es werden Grunddaten zur Zuordnung des Kunden (Kunden-ID, Name, Email), die Bestell- und Rechnungsadresse, eine Liste der bestellten Produkte mit deren Grunddaten (ID, Name, Preis, Steuerklasse, Rabatte und Boni), sowie Angaben zur Zahlungsweise berücksichtigt. Eine Aufstellung der einzelnen Attribute, die zu jeder Bestellung erfasst werden, sind der Tabelle A.4 im Anhang A.1, Seite XIX zu entnehmen.

Export von Produktdaten Produktdaten werden mit dem Befehl `Export` und dem Parameter `-products` (Kurzschreibweise `-p`) exportiert. Bei Angabe des Parameter `-since` werden nur Produkte exportiert, die nach dem angegebenen Zeitpunkt angelegt oder geändert wurden. Andernfalls werden alle gespeicherten Produkte exportiert. Zu den Produktdaten gehören die Angaben, unter welchen Katalogkategorien und Websites das Produkt gezeigt werden soll, desweiteren Angaben zum Preis und zur Lagerhaltung des Produkts, sowie allgemeine Produktinformationen und Angaben zur Suchmaschinenoptimierung der Produktdetailseite. Die Auflistung der einzelnen Attribute ist in Tabelle A.3 im Anhang A.1, Seite XVII zu finden.

Export von Katalogdaten Katalogdaten werden mit dem Befehl `Export` und dem Parameter `-catalog` (Kurzschreibweise `-a`) exportiert. Die Angabe von `-since` hat auf diese Operation keinen Einfluss. Es wird immer der komplette Katalogbaum mit allen Kategorien und Unterkategorien ausgegeben. In der Praxis wird sich die Katalogstruktur nicht allzu oft ändern, daher wird auch diese Funktion selten benötigt und es genügt die vollständige Struktur auszugeben. Die Auflistung der einzelnen Attribute ist in Tabelle A.1 im Anhang A.1, Seite XVI zu finden.

3.8.3.2 Importfunktionen

Import von Kundendaten Kundendaten werden mit dem Befehl `Import` und dem Parameter `-customers` (Kurzschreibweise `-c`) importiert. Dabei wird für jeden Datensatz geprüft, ob er schon in Magento Commerce existiert. Ist dies der Fall, so kann durch Angaben in der zentralen Einstellungsdatei festgelegt werden, ob die Daten von Magento überschrieben oder beibehalten werden sollen. Als Identifikationsmerkmal, ob ein Datensatz bereits existiert, wird die Emailadresse des Kunden verwendet. Eine Emailadresse kann nur einem Kunden zugeordnet sein. Die Emailadresse wird damit zu einem *Unique Value*, einem systemweit einmaligen Wert, und ist ein verlässliches Identifikationsmerkmal. Diese Bedingung stellt auch Magento Commerce selbst bei der Erfassung von Kundendaten.

Import von Produktdaten Produktdaten werden mit dem Befehl `Import` und dem Parameter `-products` (Kurzschreibweise `-p`) importiert. Das eindeutige Identifzierungsmerkmal ist hier die sog. SKU (Shop Keeping Unit), also die Artikel-Id in Magento. Anhand der SKU wird geprüft, ob ein Produkt bereits existiert. Ist das der Fall, so werden die Daten nur überschrieben, wenn dies

mit der entsprechenden Einstellung in der Konfigurationsdatei erlaubt wurde. Ist eine SKU nicht vorhanden, wird das Produkt neu angelegt. Zu beachten ist, dass immer eine Katalogkategorie benötigt wird, um das Produkt im Katalog einzuordnen.

Import von Katalogdaten Katalogdaten werden mit dem Befehl `Import` und dem angehängten Parameter `-catalog` (Kurzschreibweise `-a`) importiert. In der Importdatei kann der komplette Katalog enthalten sein, oder auch nur Teile davon. Zu jeder Katalogkategorie gehört eine eindeutige ID, anhand derer unterschieden wird, ob es diese Kategorie bereits gibt. Ist das der Fall, werden die dazu gehörigen Daten nur dann überschrieben, wenn dies in der zentralen Einstellungsdatei erlaubt ist. Wird für einen Eintrag keine Id gefunden, so wird der Eintrag neu angelegt. Beim Import der Katalogdaten sollte besonderes Augenmerk auf die Korrektheit aller Daten gelegt werden. Der Katalog und die Angaben darin sind im Zwiefel eher von Magento als durch die Warenwirtschaft zu bestimmen, da das System diese zur korrekten Funktionalität benötigt. Ein automatischer Import von Katalogdaten ist daher in den allermeisten Fällen nicht nötig.

3.8.3.3 Benachrichtigung und Protokollierung

Für den Betrieb in einem Produktivsystem, also bei der realen Verwendung mit echten Daten, ist es wichtig, stets zu wissen, ob die Anwendung fehlerfrei arbeitet. Tritt ein Fehler auf, so muss dieser zum Einen möglichst detailliert beschrieben und gespeichert werden. Zum Anderen muss ein zuständiger Entwickler oder verantwortlicher Mitarbeiter über den Fehler informiert werden, um zeitnah reagieren und den Fehler beheben zu können. Die Integrationslösung beinhaltet daher eine Funktion zur Protokollierung aller Ausgaben in einer Logdatei. Außerdem werden auftretende Fehler in einer separaten Fehlerdatei gespeichert. Zudem kann im Fehlerfall eine Email mit der Fehlerbeschreibung an eine konfigurierbare Adresse gesendet werden.

3.8.3.4 Konfigurierbarkeit

Die Integrationslösung ist in wesentlichen Punkten konfigurierbar. Insbesondere sind die Umformungsregeln der Daten leicht anpassbar, um andere WaWi's anstelle von ALEA ankoppeln zu können. Diese Umformungsregeln sind in sog. *Mappingdateien* gespeichert. Es existiert jeweils eine Mappingdatei für

eine Warenwirtschaft, sowie eine Mappingdatei für einen Online-Shop. Im Umfang der Arbeit wurden jedoch nur Dateien für Magento Commerce und ALEA Commerce Suite erstellt. Weitere Informationen zur Verwendung und zur Anpassung der Mappingdateien sind in Abschnitt 4.6, S. 117 zu finden.

Neben den Mappings sind in einer zentralen Einstellungsdatei viele Einstellungen für den Betrieb der Integrationslösung hinterlegbar. Das umfasst Benachrichtigungs- und Protokollierungsoptionen ebenso, wie technische Angaben, bspw. Zugangsdaten für die Schnittstelle zu Magento oder Angaben zu einem Mailserver, der das Versenden von Benachrichtigungsmails übernimmt. In der Konfigurationsdatei werden auch bestimmte Regeln festgelegt, wie das Skript bei Integrationsaufgaben mit konkurrierenden Daten umgehen soll. So ist es z.B. einstellbar ob Informationen zu Kundendatensätzen, die im Shop bereits vorhanden sind, durch neue Daten aus der Warenwirtschaft überschrieben werden sollen. Und schließlich sind auch die Speicherorte der Austauschdateien für und von ALEA Commerce Suite sowie der zugehörigen Mappingdatei konfigurierbar.

4 Beschreibung der implementierten Softwarelösung

In diesem Kapitel wird die Umsetzung der Softwarelösung näher beleuchtet, die als Lösung für den in Kapitel 3 dargestellten Integrationsfall entwickelt wurde. Um den Rahmen der Arbeit nicht zu sprengen, soll an dieser Stelle jedoch nicht auf die Programmierung im Detail eingegangen werden. Vielmehr soll ein Überblick gegeben werden, wie die Softwarelösung im Prinzip arbeitet, welchen Funktionsumfang sie besitzt und welche Maßnahmen nötig sind, um eine andere als die verwendetete Warenwirtschaft ansprechen zu können, wie es im Integrationsszenario (vgl. Abschnitt 3.1, S. 60) beschrieben ist. Desweiteren werden die technischen Voraussetzungen beschrieben, die gegeben sein müssen, damit die Integrationslösung funktionieren kann, sowie die eingesetzten Technologien vorgestellt, die den Datenaustausch und die Automatisierung der Datenoperationen gewährleisten.

4.1 Technische Voraussetzungen

Zunächst werden die technischen Resourcen benötigt, um die beiden Integrationskomponenten Magento Commerce und ALEA Commerce Suite unabhängig voneinander zu betreiben. Die entsprechenden Anforderungen wurden bereits in den Abschnitten zu den Kompontenten beschrieben (vgl. Abschnitt 3.3.1, S. 70 zu Magento Commerce und Abschnitt 3.3.2, S. 71 zu ALEA Commerce Suite). Sie können wie folgt zusammengefasst werden:

Für Magento Commerce werden benötigt[1]:

- ein dedizierter Server, der ausreichend Prozessor- und Speicherleistung und ein für den Servereinsatz optimiertes Linux-Betriebssystem zur Verfügung stellt, und über eine Verbindung mit hoher Bandbreite für Requests aus dem Internet verfügbar ist.
- der Webserver-Dienst `Apache` aus der Linie 2.2.x mit aktiviertem Modulen für die Verarbeitung von PHP und mod_rewrite
- `PHP` in Version 5.2.0 (seit Magento 1.3.0) oder höher, sowie die folgenden Erweiterungen (Extensions):
  - `PDO_MySQL`: PHP-Adapter für die Verwendung von MySQL-Datenbanken
  - `simplexml`: Bibliothek zum einfachen Verwenden von XML-Dateien
  - `mcrypt`: stellt diverse Verschlüsselungsalgorithmen bereit
  - `hash`: Bibliothek für verschiedene Hashingalgorithmen

[1] aktuelle Systemanforderungen unter http://www.magentocommerce.com/system-requirements, letzter Aufruf am 06.05.2009

  - `GD`: Funktionen zum Erzeugen und Manipulieren von Grafiken
  - `DOM`: Bibliothek zur Manipulation und Erzeugung von DOM-Dokumenten, z.B. XML oder XHTML
  - `iconv`: Modul zur Konvertierung von Dateien in verschiendene Zeichensätze, abhängig vom Betriebssystem
  - `SOAP`: SoapClient und SoapServer zur Verwendung eines auf SOAP basierenden Web Service. Die SOAP-Extension für PHP muss nur dann installiert sein, wenn die Magento Core API über SOAP verwendet werden soll.
- Das Datenbanksystem MySQL 4.1.20 oder höher. Zu empfehlen ist aber der Einsatz einer aktuellen MySQL-Version aus der Entwicklungslinie 5.1.x. Die verwendete Version muss außerdem das Speichersystem InnoDB unterstützen.

Für ALEA Commerce Suite werden die folgenden Systeme und Software benötigt:

- ein Serverrechner entweder Power System mit IBM OS/400 bzw. IBM i Betriebssystem, oder Intel/x86 Architektur mit Linux-Betriebssystem, und Netzwerkanbindung im Unternehmen,
- ein DBMS zur Speicherung der anfallenden Daten, je nach Betriebssystem des Serverrechners, z.B. IBM DB/2,
- eine Java VM, sowie
- ein Application Server, in dem ALEA Commerce Suite betrieben wird, z.B. den IBM WebSphere Application Server. Dieser ist im Lieferumfang bereits enthalten.

Wie im Abschnitt 3.8.1, S. 93 beschrieben, ist die Integrationslösung in erster Linie für einen Betrieb auf dem lokalen System von Magento Commerce ausgelegt. Daher sind auch die technischen Anforderungen der Integrationslösung weitgehend deckungsgleich mit denen von Magento Commerce. Benötigt werden ebenfalls PHP 5.2.0 oder höher mit den Erweiterungen `simplexml`, `DOM` und `SOAP`. Als zusätzliche Erweiterung wird das PEAR-Paket `Console_CommandLine`[2] benötigt. Dieses stellt die Funktionen zur Interaktion des Skripts mit der Kommandozeile des Betriebssystems bereit. Magento Commerce selbst muss in der Version 1.3.0 oder höher vorliegen, wenn der Appmode verwendet werden soll. Kommuniziert die Anwendung dagegen über die Magento Core API, so funktioniert auch Magento ab Version 1.2.0 ohne Einschränkungen. Prinzipiell sollte jedoch immer die aktuellste verfügbare

[2] http://pear.php.net/package/Console_CommandLine, letzter Aufruf am 06.05.2009

stabile Version von Magento verwendet werden, da in jedem neuen Release eine ganze Reihe von Fehlern behoben wird, was unter Umständen existierende Probleme mit älteren Versionen beseitigt.
Im Gegensatz zu Magento lässt sich die Integrationslösung auch unter einem anderen als einem Linux-Betriebssystem betreiben und es wird für die Integrationslösung auch keine Datenbank benötigt. Auch der Einsatz eines anderen Webservers als Apache ist prinzipiell möglich. Diese Flexibilität ermöglicht den Einsatz der Integrationslösung auf sehr verschiedenen Systemen, wenn die Lösung mit dem Magento Appmode nicht möglich sein sollte.

Um die Integrationslösung automatisieren zu können, sodass sie regelmäßig bestimmte Integrationsoperationen ausführt, kann unter Linux-Betriebssystemen der Befehl `crontab` verwendet werden. `crontab` und der dazugehörige Dienst `cron` gehören zum Standardumfang einer jeden Linux-Distribution. Abschnitt 4.3, S. 108 befasst sich genauer mit der Verwendung von `crontab` im Rahmen des vorliegenden konkreten Integrationsfalls. Wird die Integrationslösung unter einem anderen Betriebssystem verwendet, müssen die dort vorhandenen Automatisierungsdienste analog verwendet werden. Auf einem Windows-Server ist dies z.B. mit dem Taskplaner umsetzbar.

Als letzte Komponente wird eine Anwendung benötigt, die XML-Austauschdateien zwischen dem lokalen System der Warenwirtschaft und dem lokalen System der Integrationslösung transportieren kann. Im Rahmen der Arbeit wird das OpenSource-Programm `rsync` in Verbindung mit der →Secure Shell (SSH) verwendet. Beide sind in Linux-Betriebssystemen entweder schon standardmäßig installiert (bzw. OpenSSH als OpenSource-Implementierung von SSH) oder lassen sich mit einem Kommandozeilenbefehl installieren. Für viele andere Betriebssysteme existieren Übertragungen (sog. Ports) von `rsync`, wie u.a. `Delta Copy`[3] für Windows-Server. Mit der konkreten Verwendung von `rsync` im Rahmen des hier behandelten Integrationsfalls befasst sich der Abschnitt 4.2 im Anschluss.

4.2 Datenaustausch mit der Warenwirtschaft

Aus dem Integrationsszenario geht hervor, dass die Warenwirtschaft völlig autark von der Integrationslösung und dem Online-Shop angelegt ist und auch sein muss. Daraus lässt sich die Notwendigkeit ableiten, dass die zum Austausch der für die Integrationsoperationen nötigen Daten erzeugten Dateien in

[3] http://www.aboutmyip.com/AboutMyXApp/DeltaCopy.jsp, letzter Aufruf am 06.05.2009

geeigneter Art und Weise physisch vom lokalen System der Warenwirtschaft ins lokale System der Integrationslösung übertragen werden müssen, und umgekehrt. Diese Übertragung muss für den praktischen Einsatz den folgenden Ansprüchen genügen: Sie muss *sicher verschlüsselt* erfolgen, besonders wenn die Übertragung über ungesicherte Netzwerke oder das Internet erfolgt. Die Übertragung muss bei Bedarf *von beiden Seiten ausgelöst* werden können, da immer nur der Produzent einer neuen Austauschdatei wissen kann, dass diese existiert. Und schließlich sollte die Übertragung möglichst *schnell* erfolgen, um die benötigten Zeiten zum Austausch der Daten möglichst gering zu halten. Denn je länger der Austausch dauert, desto größer muss das Mindestzeitintervall sein, in dem Integrationsoperationen durchgeführt werden können.

Im Rahmen der Umsetzung der Integrationslösung in dieser Arbeit wurde für den Datenaustausch das OpenSource-Tool `rsync`[4] verwendet. `rsync` ist ein Netzwerkprotokoll und Programm, das speziell für die Synchronisation von Dateien über jegliche Art von Netzwerken konzipiert ist. Das Besondere an `rsync` ist, dass es nur die Teile von Dateien überträgt, die auf dem Zielsystem nicht vorhanden sind, und sich auch nicht aus Teilen der Zieldatei zusammensetzen lassen. Dadurch wird bei teilweise übereinstimmenden Daten nur ein Bruchteil der Datenmenge über das Netzwerk übertragen. `rsync` arbeitet nach dem Client/Server-Prinzip. Bei einer Synchronisationsoperation sendet der Client zunächst Prüfsummen der zu übertragenden Daten an den Server, der anhand dieser entscheidet, welche Teile der Daten vom Client gesendet werden sollen. Anschließend fügt der Server aus den empfangenen und den bereits vorhandenen Teilen die Daten zusammen und speichert sie auf seinem lokalen System. Daraus folgt, dass die Syncronisation immer nur in einer Richtung – vom Client zum Server – funktioniert. Sollen Daten auch in die entgegengesetzte Richtung übertragen werden, so müssen die Rollen von Server und Client auf den Systemen getauscht werden. Beide Systeme müssen daher sowohl einen Server als auch einen Client besitzen, und beide Systeme müssen über eigene Regeln verfügen, nach denen sie Übertragungsoperationen auslösen. Da `rsync` bereits vorhandene Inhalte in den Zielverzeichnissen nicht beibehält, sondern nur syncronisiert, muss für beide Richtungen jeweils ein eigenes Verzeichnis genutzt werden, damit keine Dateien versehentlich überschrieben oder gelöscht werden.

`rsync` kann durch eine Vielzahl von Parametern sehr flexibel für alle Arten von Datensyncronisationen verwendet und in seinem Verhalten angepasst werden. Eine für die Integration wichtige Eigenschaft ist, dass es bei Angabe des entsprechenden Parameters (`-t`) den Zeitstempel einer Datei bei der Syncroni-

[4] http://www.samba.org/rsync/, letzter Aufruf am 06.05.2009

sation nicht verändert. Das ermöglicht es, den Zeitpunkt der letzten Änderung an der Datei, z.B. durch die erstellende Warenwirtschaft, auch nach der Syncronisation auf das System der Integrationslösung zu erhalten. So kann die Integrationslösung durch das Auslesen des Zeitstempels feststellen, ob die Datei seit ihrem letzten Zugriff durch die Warenwirtschaft geändert wurde. Die Dateioperationen während der Syncronisation bleiben damit für die Integrationskomponenten transparent und die Dateien und deren Zeitstempel können verwendet werden, als wären sie von einer lokalen Anwendung erzeugt worden. Auf alle Einstellungen einzugehen, ist im Rahmen dieser Arbeit nicht möglich. Für eine weitergehende Beschäftigung mit `rsync` sei deshalb an dieser Stelle auf die sehr ausführliche Dokumentation auf der Internetseite des Projekts verwiesen[5].

`rsync` kann für die Übertragung das Verschlüsselungsprotokoll → SSH verwenden. Damit ist eine sichere Übertragung von Daten auch über ungesicherte Netzwerke problemlos und bequem möglich. Unter Linux wird bei der Installation von `rsync` standardmäßig ein SSH-Server mit installiert. Die spätere Verwendung gestaltet sich sehr einfach: Über einen Kommandozeilenbefehl kann eine Syncronisationsoperation über eine SSH-Verbindung ausgeführt werden. Die Tabelle 4.1 zeigt ein Beispiel für einen solchen Befehl unter Linux.

```
1 rsync -e ssh /var/data/Export/ user:password@192.168.0.1:/var/data/Import/
```

Tabelle 4.1: Beispiel für einen Aufruf von rsync

Dieser Befehl kopiert den Inhalt des lokalen Verzeichnis `/var/data/Export/` auf das entfernte System, das über die IP-Adresse `192.168.0.1` erreichbar ist, und dort ins Verzeichnis `/var/data/Import/`. Zur Übertragung wird eine SSH-Verbindung aufgebaut, und am SSH-Server des entfernten Systems werden zur Anmeldung die Benutzerdaten `user` und `password` verwendet. Zu beachten ist dabei auch, dass der verwendete Benutzer für das angegebene Verzeichnis Lese- und Schreibrechte besitzen muss.

Zum Auslösen einer Syncronisationsoperation genügt also der Aufruf eines Kommandozeilenbefehls auf dem System, dass die Syncronisation auslösen möchte, z.B. weil es gerade eine neue Austauschdatei erzeugt hat. Es gibt auf dem System zwei prinzipielle Möglichkeiten, die Syncronisation auszulösen: Zum Einen kann der Befehl nach der Erzeugung der Austauschdateien mit den neuen Daten vom erzeugenden Programm aufgerufen werden. Zum Anderen kann der Befehl aber auch automatisiert regelmäßig vom System selbst durch

[5] http://rsync.samba.org/ftp/rsync/rsync.html, letzter Aufruf am 07.05.2009

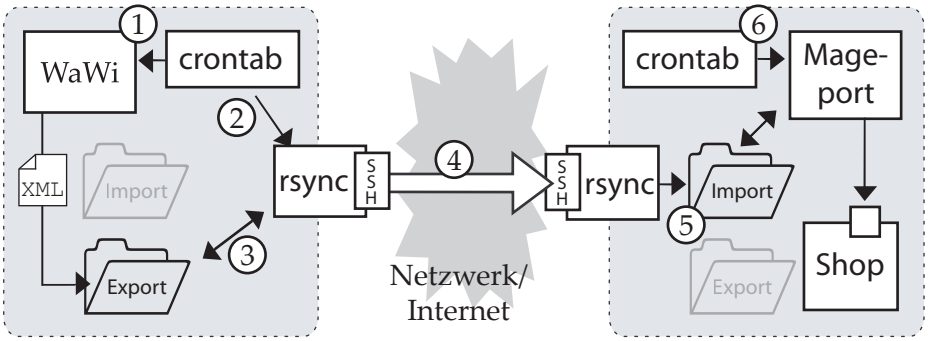

Abbildung 4.1: Ablauf der Syncronisationsoperationen mit rsync

einen *Cronjob* aufgerufen werden (s. Abschnitt 4.3, S. 108). Die Entscheidung für eine der beiden Vorgehensweisen ist je nach dem konkreten Integrationsfall zu treffen. In jedem Fall müssen auf beiden Systemen `rsync` und ein SSH-Server vorhanden sein und Regeln für den Aufruf der Syncronisationsoperationen definiert werden.

Zur Umsetzung des konkreten Integrationsszenarios wurde auf beiden Systemen die Variante über einen regelmäßigen Befehlsaufruf durch das Betriebssystem angewendet. Es wurde jeweils ein Cronjob eingerichtet, der die Austauschdateien zwischen den Systemen syncronisiert. Für den Austausch von Dateien von der Warenwirtschaft zur Integrationslösung wurde ein Intervall von 60 Minuten bestimmt, da hier weniger oft neue Daten zu erwarten sind. Für den Austausch in der Gegenrichtung wurde ein Intervall von 15 Minuten angesetzt, vor allem damit Daten zu neuen Bestellungen möglichst umgehend in die Warenwirtschaft aufgenommen werden können. Die Abbildung 4.1 verdeutlicht den schematischen Ablauf der Syncronisation ausgehend vom System der Warenwirtschaft: Zuerst wird per `crontab` in der warenwirtschaft eine Exportoperation ausgelöst, sodass eine aktuelle Austauschdatei im dafür vorgesehenen Exportordner abgelegt wird (1). Anschließend ruft die `crontab` `rsync` auf (2), das die Austauschdatei aus dem Exportordner (3) über eine SSH-Serverschnittstelle über das Internet (4) zum `rsync` des Integrationspartner überträgt. Dieses legt die XML-Datei in den Importordner seines lokalen Systems (5). Wird auf diesem System nun per `crontab` das Mageport-Skript aufgerufen (6), wird es die aktuelle XML-Datei verwenden, um die Datein in den Shop zu importieren.

4.3 Automatisierung

Im praktischen Betrieb der Integrationslösung soll die Integration der Daten zwischen der Warenwirtschaft und dem Online-Shop ohne menschliche Aktivität erfolgen. Für die Durchführung von Integrationsoperationen ist jedoch der Aufruf und die Ausführung von Programmen nötig, die verschiedene Teilaufgaben der Gesamtoperation übernehmen. Für die Übertragung von Produktdaten aus der Warenwirtschaft in den Shop sind bspw. die folgenden Schritte nötig: Zuerst muss die Wartenwirtschaft eine XML-Austauschdatei erzeugen, die die zu übertragenden Daten enthält, und diese in einem dafür vorgesehenen Ordner speichern. Anschließend muss `rsync` (vgl. Abschnitt 4.2, S. 104) aufgerufen werden, um die XML-Datei vom lokalen System der Warenwirtschaft ins lokale System der Integrationslösung zu übertragen. Schließlich muss die Integrationslösung aktiviert werden, welche die XML-Datei verarbeitet und die Daten in den Online-Shop überträgt. Für eine solche Integrationsoperation sind also drei automatisierte Programmaufrufe auf zwei verschiedenen Systemen notwendig.

Zur Realisierung automatischer Programmaufrufe bringen verschiedene Server-Betriebssysteme verschiedene Funktionalitäten mit. In Linux (und anderen von Unix abgeleiteten Betriebssystemen wie IBM OS oder Apple Mac OS) gibt es zu diesem Zweck den Dienst `cron`, der dafür zuständig ist, wiederkehrende Befehle zu festgelegten Zeitpunkten selbständig auszuführen. Ein solcher Befehl wird auch als *Cronjob* bezeichnet. Ein Cronjob beinhaltet die Angabe von Tag und Zeit der Ausführung und den auszuführenden Befehl. Cronjobs werden in der sog. `crontab` gespeichert, einer Tabelle von Cronjobs, auf deren Grundlage der Dienst `cron` die Befehle zur richtigen Zeit automatisch ausführt. Statt der Angabe einzelner Ausführungszeiten ist auch die Angabe von Zeitintervallen möglich, also z.B. die Ausführung alle 15 Minuten. Cronjobs werden auf Benutzerbasis angelegt, sodass für jeden Benutzer des Systems eine eigene `crontab` existiert. Die Befehle werden von `cron` dann mit den Rechten des jeweiligen Benutzers ausgeführt. Das ermöglicht eine fein differenzierte Ausgestaltung der möglichen Operationen auf Benutzerbasis. Erstellte Cronjobs werden auch dann wie vorgesehen mit den Rechten des Benutzers ausgeführt, wenn dieser nicht am System angemeldet ist. Die Syntax der `crontab` wird in Tabelle 4.2 dargestellt und ist relativ leicht zu lesen. Kommentarzeilen werden mit dem Rautezeichen (`#`) eingeleitet, der Asterisk (`*`) steht für „alle möglichen Werte“, Listen und Wertebereiche sind ebenfalls erlaubt.

Zum besseren Verständnis seien die Angaben aus der Tabelle 4.2 an dieser Stelle übersetzt aufgeführt: Die erste Zeile ist eine Kommentarzeile. Sie be-

```
1 # Minute   Stunde     Tag   Monat   Wochentag   Befehl
2   30       2          *     *       1-5         /usr/bin/dosomething.sh
3   /10      *          *     *       *           /usr/bin/update.sh
4   0        10,15,18   1     4       *           cp /var/aprilfool/* var/messages
```

Tabelle 4.2: Syntax von Crontab-Einträgen

nennt die Bedeutung der Spalten und wird von `cron` nicht interpretiert. In Zeile 2 ist definiert, dass die Datei `/usr/bin/dosomething.sh` immer Montags bis Freitags um zwei Uhr 30 nachts aufgerufen wird. Die Zeile 3 definiert einen Cronjob, der alle zehn Minuten die Datei `/usr/bin/update.sh` aufruft. Mit Zeile 4 wird festgelegt, dass am ersten April jeweils um zehn Uhr, um 15 Uhr und um 18 Uhr alle Inhalte des Ordners `/var/aprilfool` in den Ordner `/var/messages` kopiert werden.

Für die Umsetzung der Integrationslösung für den konkreten Integrationsfall dieser Arbeit werden Cronjobs sowohl auf dem System der Warenwirtschaft, als auch auf dem System des Online-Shops verwendet. Die Tabellen 4.3 und 4.4 zeigen beispielhaft, wie die Automatisierung der einzelnen Teilaufgaben für Integrationsoperationen organisiert sein kann. Die Befehle zur Interaktion mit Alea in diesem Beispiel sind fiktiv.

```
1 # Export von neuen Daten für Shop jede Stunde
2 0  *  *  *  *  alea export --new
3 # Export aller Daten für Shop einmal pro Tag nachts
4 30  2 *  *  *  alea export --all
5 # Austauschdatei mit Shop jede Stunde syncronisieren
6 1  *  *  *  *  rsync -az -e ssh /var/data/Export
7                      user:password@onlineshop:/var/data/Import
8 # Import von neuen Daten aus dem Shop alle 10 Minuten
9 /10  *  *  *  *  alea import -a -d /var/data/Import
```

Tabelle 4.3: Cronjobs auf dem System der Warenwirtschaft

```
1 # Export von Bestellungen für WaWi alle 10 Minuten
2 /10  *  *  *  *  /etc/mageport/export.sh --orders
3 # Austauschdatei alle 10 Minuten mit WaWi syncronisieren
4 /10  *  *  *  *  rsync -az -e ssh /var/data/Export
5                      user:password@wawi:/var/data/Import
6 # Import von neuen Daten aus der WaWi jede volle Stunde
7 2  *  *  *  *  /etc/mageport/import.sh --all
```

Tabelle 4.4: Cronjobs auf dem System der Integrationslösung

Auf dem System der Warenwirtschaft gibt es in diesem Fall zwei Cronjobs, die die Erzeugung der XML-Austauschdateien starten. Durch die Übergabe

geeigneter Parameter an den Erzeugungsbefehl kann zwischen neuen Daten, die seit dem letzten Export erfasst wurden, und allen vorhandenen Daten unterschieden werden. Die neuen Daten werden – so vorhanden – zu jeder vollen Stunde exportiert. Zusätzlich wird einmal täglich der komplette Datenbestand exportiert, damit dem Shop und der Warenwirtschaft konsistente Daten zugrunde liegen. Dies geschieht optimalerweise in der Nacht, wenn die Systeme am wenigsten belastet sind. Eine Minute nach jeder vollen Stunde werden die Austauschdateien im Export-Ordner per `rsync` auf das System der Integrationslösung übertragen. Die eine Minute Verzögerung soll sicherstellen, dass der vorherige Befehl zur Erzeugung der aktuellen Austauschdateien bereits fertiggestellt ist. Zusätzlich wird alle 10 Minuten der Import in die Warenwirtschaft gestartet, der – so vorhanden – neue Daten aus dem Onlineshop in die Warenwirschaft aufnimmt.

Auf dem System der Integrationslösung sind in diesem Beispiel drei Cronjobs definiert. Der erste exportiert alle zehn Minuten – so vorhanden – neue Bestellungen und speichert sie in einer XML-Austauschdatei. Der zweite Cronjob nutzt dann wiederum `rsync` um diese Datei zum System der Warenwirtschaft zu übermitteln. Auch dies geschieht alle zehn Minuten. Der dritte Cronjob importiert zwei Minuten nach jeder vollen Stunde neue Daten aus der Austauschdatei, die eine Minute vorher vom System der Warenwirtschaft aus hinterlegt wurde, in den Shop. Damit ist der Informationskreislauf geschlossen. Durch die Definition anderer Cronjobs mit anderen Befehlsaufrufen und deren entsprechende zeitliche Abstimmung können die Informationsflüsse anforderungsgerecht und individuell ausgestaltet werden.

4.4 Aufbau der Integrationslösung

Im Abschnitt 3.8.1, S. 93 wurde dargelegt, aus welchen Gründen die Integrationslösung als eigenständiges Skript, und somit als eigenständige Anwendung umgesetzt wurde. In diesem Abschnitt soll nun der Aufbau und die Bestandteile der Umsetzung betrachtet werden. Als Bezeichnung für die Integrationslösung wurde der Name *Mageport* gewählt, eine verkürzte Kombination aus den Begriffen *Magento* und *Import*, bzw. *Export*. Mit diesem Namen wird die Integrationslösung im Folgenden benannt.

Die Abbildung 4.2 zeigt einen Screenshot der Liste aller zu Mageport gehörenden Dateien. Mageport ist zunächst eine in Ordnern strukturierte Sammlung von Dateien, die im Zusammenspiel die durch das Integrationsszenario und den in Abschnitt 3.8.3, S. 96 definierte Funktionalität bereitstellen. Die allermeisten

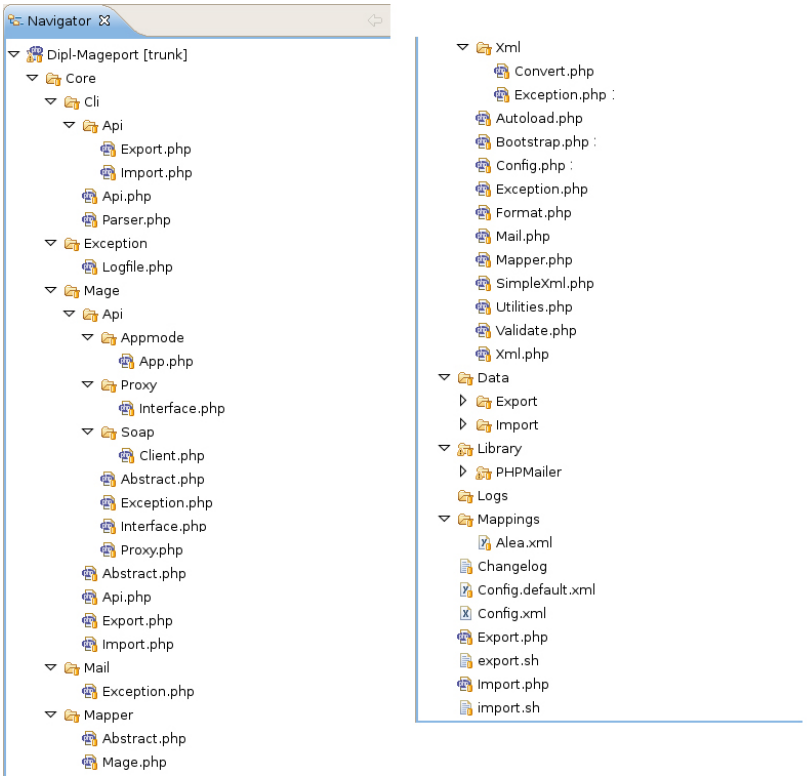

Abbildung 4.2: Dateistruktur der Integrationslösung

Dateien sind →PHP Dateien. Sie sind leicht an der Endung `.php` zu erkennen und enthalten PHP-Skriptcode. Da Mageport objektorientiert umgesetzt wurde, enthält jede PHP-Datei eine gleichnamige Klasse, die Funktionalitäten für einen bestimmten Zweck zur Verfügung stellt. Neben den PHP-Dateien enthält Mageport noch einige →XML-Dateien. Diese dienen entweder der Speicherung von Konfigurationseinstellungen, oder es handelt sich um Mapping- oder Austauschdateien, die die zu behandelnden Daten modellieren, oder beinhalten. Den dritten Dateityp bilden zwei Shell-Dateien mit der Endung `.sh`. Diese dienen lediglich dazu, den Aufruf von Mageport-Funktionen über die Kommandozeile auf Linux-Betriebssystemen etwas bequemer zu gestalten und haben keine darüber hinausgehende eigene Funktion.

Der Ordner `Core` beinhaltet alle Klassen, die in ihrer Gesamtheit die Funktionalität von Mageport implementieren. Er dient gleichzeitig als Namensraum für alle Mageport-Klassen. Die Implementierung verwendet die →PEAR Codingstandards[6] und so sind auch die Klassennamen eng mit der Ordnerhierarchie verknüpft. Die Datei `Core/Cli/Api/Import.php` beinhaltet also die Klasse `Core_Cli_Api_Import`, welche – wie der Klassenname bereits andeutet –

[6] http://pear.php.net/manual/en/standards.php, letzter Aufruf am 08.05.2009

die erlaubten Parameter definiert, die für den Import beim Aufruf von Mageport über die CLI-Schnittstelle verwendet werden können. Innerhalb des Core-Ordner sind die grundlegend benötigten Komponenten zu erkennen: die zentrale Klasse `Core_Bootstrap`, die als Basisklasse den Gesamtablauf der Anwendung und und die Koordinierung der einzelnen Komponenten umsetzt, die Funktionalitäten der Kommandozeilenschnittstelle (Ordner `Core/Cli`), Methoden zur Behandlung und zum Auszeichnen von Fehlern (`Core_Exception` und der Ordner `Core/Exception`), das Versenden von Benachrichtigungsmails (`Core_Mail`), sowie Funktionen zum Auslesen, Erzeugen und Umwandeln von XML-Dateien (`Core_Xml` und `Core/Xml`). Der Ordner `Core/Mage` beinhaltet alle Klassen, die für die Kommunikation mit Magento Commerce zuständig sind. Neben den Klassen, die die einzelnen Integrationsoperationen beherbergen (`Core_Mage_Import` und `Core_Mage_Export`), sind dort auch die Schnittstellen zur Magento Core API (`Core_Mage_Api_Soap_Client`) und zum Magento Appmode (`Core_Mage_Api_Appmode_App`) angesiedelt (vgl. Abschnitt 3.5.3, S. 83; bzw. Abschnitt 3.5.4, S. 87). `Core_Mage_Import` und `Core_Mage_Export` verwenden diese jeoch nicht direkt, sondern nutzen stattdessen eine Proxy-Klasse (`Core_Mage_Api_Proxy`). Diese entscheidet, welche der beiden Schnittstellen je nach Konfiguration verwendet werden soll. Die Import- und Exportklassen müssen daher keine Kenntnis davon haben, welche Schnittstelle gerade zu verwenden ist, und können für beide Schnittstellen in gleicher Weise verwendet werden. Im Ordner `Core/Mapper` schließlich befinden sich die Klassen, die alle Datenattribute definieren, die die Integrationslösung Mageport verarbeiten kann. Diese Attribute werden den Namen der Attribute des angebundenen Online-Shops zugeordnet. Dieselben Attribute werden auch in den Mappingdateien verwendet, die beschreiben, wie die Daten der Warenwirtschaft dargestellt und strukturiert sind. Die Klassen in `Core/Mapper` sind somit das zentrale Bindeglied zwischen den Datenmodellierungen der zu integrierenden Komponenten. Neben der abstrakten Klasse `Core_Mapper_Abstract`, die einige Funktionen bereitstellt, die für alle Mappingklassen benötigt werden, muss für jeden Online-Shop, der mit Mageport arbeiten soll, eine Mappingklasse existieren. Diese Klasse definiert eine Anzahl von Attributschlüsseln – eindeutige Bezeichner, unter denen ein Attribut in Mageport eindeutig identifiziert werden kann. Im Rahmen der Arbeit wurde nur Magento angebunden, die zugehörige Mappingklasse ist `Core_Mapper_Mage`. Die Auflistungen aller Attribute für die verschiedenen Datenobjekte, die die Mappingklassen derzeit definieren, inklusive ihrer Attributschlüssel und den entsprechenden Datenattributfeldern in Magento, befindet sich in Anhang A.1, Seite XVI.

Der Ordner `Data` beinhaltet zunächst lediglich zwei leere Unterordner, `Export` und `Import`. In diesen Ordner werden bei Verwendung der Standardeinstel-

lungen die Austauschdateien platziert. Der Order `Import` umfasst die Austauschdateien, die von der Warenwirtschaft erzeugt und mit `rsync` (vgl. Abschnitt 4.2, S. 104) hierher übertragen wurden, damit sie von Mageport in Magento Commerce eingefügt werden. Der Ordner `Export` beherbergt umgekehrt die Dateien, die von Mageport mit Daten aus Magento erzeugt wurden. Diese beiden Ordner sind also diejenigen, die mittels Cronjobs und `rsync` automatisiert mit dem System der Warenwirschaft abgeglichen werden. Der Ordner `Logs` ist standardmäßig ebenfalls leer. Hier werden die Meldungen der Anwendung gespeichert, wenn dies in der Konfiguration aktiviert ist. Auch Meldungen zu Fehlern werden in einer Datei in diesem Ordner gespeichert. Der Speicherort für Meldungen und Fehlerberichte ist jedoch in der Konfiguration änderbar. Unter Umständen wird der Ordner daher gar nicht verwendet. Der Ordner `Library` enthält verwendete Bibliotheken und Komponenten, die von externen Anbietern erstellt wurden und ohne Veränderungen wiederverwendet werden. Bei Mageport wird für den Versand von Emails das unter LGPL-Lizenz stehende OpenSource-Paket `PHPMailer`[7] benutzt. Es erledigt alle Aufgaben, die mit dem Erstellen und Versenden von Emails über Smtp-Server einhergehen. Die andere externe Komponente, die Mageport verwendet, die PEAR-Komponente `Console_CommandLine`[8], wird üblicherweise über den PEAR-Installer installiert und muss deshalb nicht im `Library`-Ordner liegen. Prinzipiell ist dies jedoch möglich. Mageport funktioniert genauso, wenn `Console_CommandLine` nicht installiert, sondern nur in das Verzeichnis `Library` entpackt wurde.

Der Ordner `Mappings` ist das Gegenstück zu den Klassen unter `Core/Mapper`. Hier befinden sich die zentralen XML-Mappingdateien, welche die Umsetzung der von Mageport verwendeten Datenattribute in das XML-Format der konkret angebundenen Warenwirtschaft realisieren. Jeder in Mageport verwendete Attributschlüssel kann in den XML-Dateien in diesem Ordner als Wert für einen XML-Tagknoten verwendet werden. Die Attributschlüssel von Mageport dienen somit als Bindeglied zwischen den in Arrayform gespeicherten Attributwerten in Magento und den in verschachtelten XML-Bäumen organisierten Attributen in den Austauschdateien der Warenwirtschaft. In Anhang A.1, Seite XVI sind die Bezeichner, sowie ihre jeweilige Entsprechung in den Datenarrays von Magento und den XML-Bäumen von ALEA Commerce Suite aufgeführt. Durch das Anlegen von weiteren XML-Dateien im Ordner `Mappings` können abweichende XML-Strukturen bzw. Anbindungen an andere Warenwirtschaftssysteme umgesetzt werden (vgl. Abschnitt 4.6, S. 117). Durch eine Einstellung in der zentralen Konfigurationsdatei wird Mageport mitgeteilt, welche XML-Mappingdatei es verwenden soll. Im Rahmen der Arbeit wurde

[7] http://phpmailer.codeworxtech.com/, letzter Aufruf am 08.05.2009
[8] http://pear.php.net/package/Console_CommandLine, letzter Aufruf am 11.05.2009

nur an der Mappingdatei für ALEA Commerce Suite gearbeitet. Diese ist als `Alea.xml` im Ordner `Mappings` gespeichert.

Zuletzt befinden sich noch einige Dateien im Hauptordner von Mageport. Diese dienen zur Konfiguration und zum Aufruf der Anwendung. Die Datei `Config.xml` ist die zentrale Einstellungsdatei, von der schon an verschiedenen Stellen die Rede war. Ihre Baumstruktur ist fest vorgegeben und muss alle Elemente, wie vorgefunden, enthalten. Die Werte jedes XML-Tags können verändert werden und bestimmen zum Einen das Verhalten der Anwendung wesentlich mit; zum Anderen liefern sie der Anwendung benötigte Informationen z.B. über Zugangsdaten zur Magento Core API oder zum Mailserver für das Versenden von Benachrichtigungsmails. Die Datei `Config.xml` ist im Auslieferungszustand von Mageport selbst nicht vorhanden. Wenn Mageport trotzdem aufgerufen wird, bricht es mit einem Hinweis auf die fehlende Datei ab. Dies ist der Fall, damit die Konfiguration als bewusster Schritt der Installation der Anwendung vollzogen wird, und es nicht durch evtl. falsch angenommene Standardkonfigurationswerte zu schwer zu lokalisierenden Problemen bei der Inbetriebnahme führt. Aus diesem Grund ist stattdessen die Datei `Config.default.xml` vorhanden. Diese Datei enthält die komplette XML-Struktur, wie sie von der Anwendung benötigt wird, lässt aber die meisten Werte leer. Diese müssen nach dem Kopieren oder Umbenennen der Datei zu `Config.xml` für das konkrete System eingefügt werden. Die bereits vorausgefüllten Felder beinhalten empfohlene Werte, die aber auch verändert werden können, wenn Bedarf danach besteht. Der Inhalt der `Config.default.xml`, sowie die Bedeutung der einzelnen Elemente und ihre Auswirkung auf die Anwendung sind in Anhang A.2, Seite XX aufgelistet.

Die Dateien `Export.php` und `Import.php` dienen zum Aufruf der Anwendung. Sie werden mit den entsprechenden Parametern aufgerufen (vgl. Abschnitt 3.8.3, S. 96), um die gewünschte Integrationsoperation mit Magento Commerce durchzuführen. Wenn Mageport auf einem Server in einem öffentlich zugänglichen Verzeichnis installiert ist, dann können diese beiden Dateien statt über die Kommandozeile auch über einen Browser aufgerufen werden. Die Dateien `export.sh` und `import.sh` schließlich dienen ausschließlich dem Aufruf über die Kommandozeile unter Linux. Sie verkürzen den zu verwendenden Befehl etwas und sind damit auf der Kommandozeile und in der `crontab` leichter zu lesen und einzugeben.

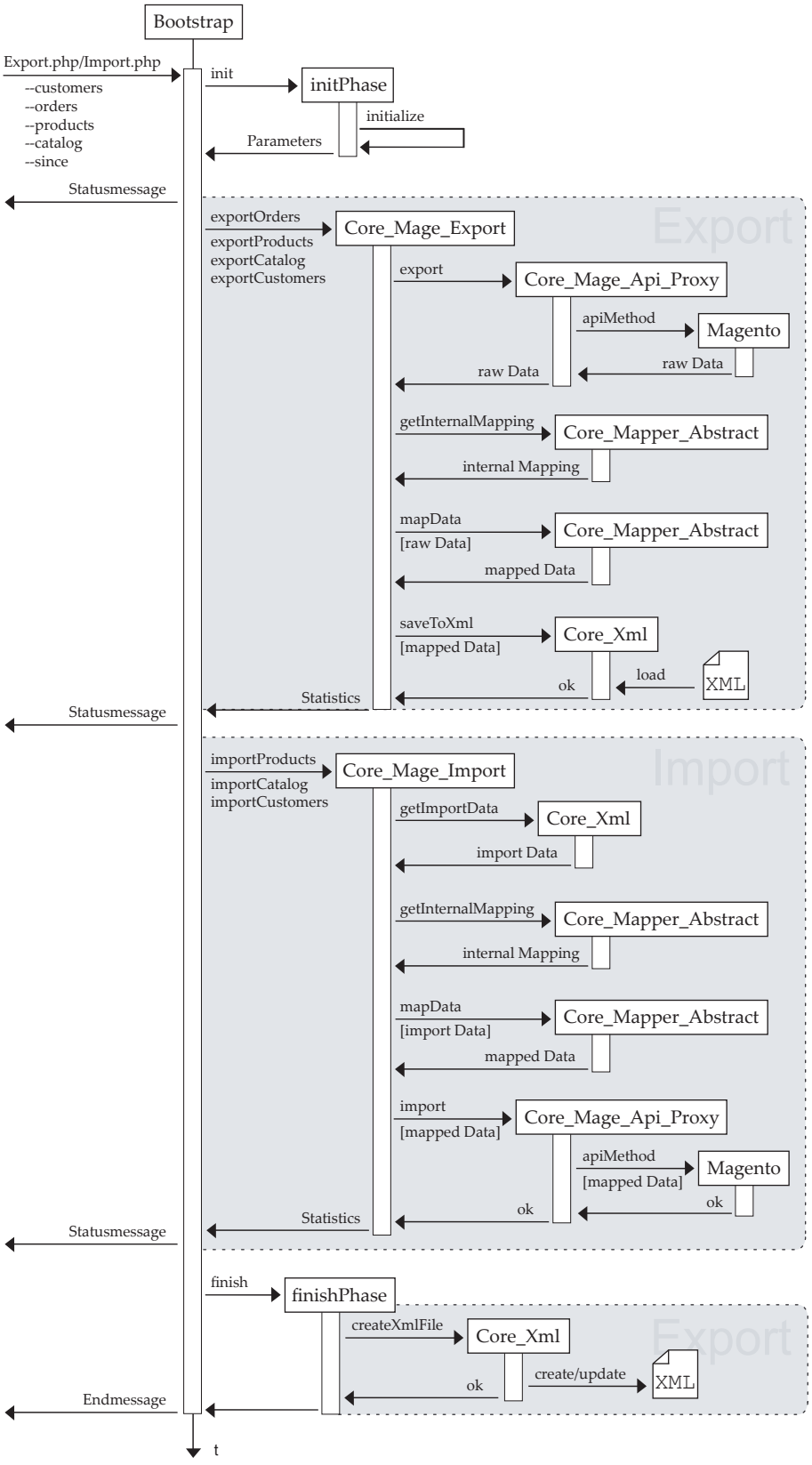

Abbildung 4.3: schematischer Programmablauf von Mageport

4.5 Prinzipieller Programmablauf

Die Abbildung 4.3 zeigt den Ablauf eines Programmzyklus, wie er beim Aufruf von Mageport durchlaufen wird, als UML-Sequenzdiagramm. Der Aufruf des Befehls `export.sh -orders` bewirkt in Mageport die in der Abbildung dargestellten Aktionen und Abläufe: Die Datei `export.sh` ruft die Datei `Export.php` auf und übergibt ihr den erhaltenen Parameter `-orders`; diese wiederum lädt die zentrale Klasse `Core_Bootstrap`, die den weiteren Ablauf steuert. Zuerst werden verschiedene Initialisierungsmethoden durchgeführt. Diese prüfen zuerst, ob das System alle nötigen Betriebsanforderungen erfüllt, sorgen dann dafür, dass von nun an benötigte Klassen und externe Komponenten verfügbar und ansprechbar sind, und laden die Konfigurationseinstellungen aus der `Config.xml`. Anschließend wird `Console_CommandLine` aufgerufen, das die verwendeten Parameter aus der Nutzereingabe ermittelt, auf Zulässigkeit prüft und an die Bootstrap-Klasse zurück gibt. Sind bis hierher keine Fehler aufgetreten, geht die Anwendung davon aus, dass alle nötigen Voraussetzungen für die Durchführung der Integrationsperationen vorliegen. Sie gibt nun einen kurzen Statusbereicht aus und ruft nacheinander für jeden korrekt erhaltenen Nutzerparameter die entsprechende Subroutine auf, die den inneren Ablauf einer einzelnen Operation steuert. Die Abbildung 4.3 zeigt jeweils den Ablauf einer Import-, und einer Exportoperation. In der Praxis treten diese beiden Operationstypen jedoch nicht gemeinsam in einem Anwendungszyklus auf, da die möglichen Eingangsbefehle `import.sh` und `export.sh` bereits eine Entscheidung für einen der Typen implizieren. Als Ergebnis jeder Subroutine wird ein kurzer Statusbericht zurück an die Bootstrap-Klasse gegeben, den diese als Information für den Nutzer ausgibt, bevor die nächste Subroutine gestartet wird. Nach Abschluss aller Subroutinen wird noch eine `finish` Routine durchgeführt. Diese ist z.B. bei Exportoperationen für die Erstellung der XML-Austauschdatei aus den erhaltenen Daten zuständig. Im Anschluss beendet die Bootstrap-Klasse die Ausführung mit einem Hinweis auf die Beendigung der Operationen und entfernt sich und alle Objekte der Anwendung aus dem Arbeitsspeicher.

Eine Export-Subroutine beginnt damit, dass sie die gewünschten Daten mit der Klasse `Core_Mage_Export` aus dem Magento-Shop ausliest. Sie tut das aber nicht direkt, sondern verwendet stattdessen die interne Schnittstellenklasse `Core_Mage_Api_Proxy`. Diese entscheidet dann, welche Schnittstelle zu Magento in Abhängigkeit der Konfigurationseinstellungen verwendet wird und gibt das Ergebnis zurück an die Klasse `Core_Mage_Export`. Diese wandelt die „rohen“ Ergebnisdaten zunächst in ein verschachteltes Array um, welches die Subroutine leichter verarbeiten kann. Die Subroutine erzeugt in zwei Schritten

aus dem Datenarray ein neues Datenarray, in dem nun die Datenattribute mit den eindeutigen Bezeichnern von Mageport verknüpft sind. Dieser Vorgang wird auch als *Mapping* bezeichnet. Daten werden in eine andere Repräsentationsform überführt, jedoch nicht selbst bearbeitet. Das gemappte Datenarray wird intern gespeichert und Informationen über die Anzahl der gespeicherten Elemente u.ä. werden für die Verwendung in Statusberichten an die Bootstrap-Klasse zurückgegeben. Die Erzeugung der XML-Austauschdatei wird nicht von der Export-Subroutine übernommen, da bei der Angabe mehrerer Parameter sonst die XML-Datei der ersten Subroutine von der nächsten Subroutine überschrieben würde, denn der Dateiname für die Austauschdatei ist durch Konfigurationseinstellungen vorgegeben. Zum die Exportdaten zu speichern ruft die Bootstrap-Klasse nach dem Durchlauf aller Subroutinen die Methode `finish` auf, die aus den Daten aller Subroutinen eine einzige XML-Datei erstellt.

Die Import-Subroutine beginnt stattdessen damit, die zum Import bestimmten Daten aus der XML-Importdatei auszulesen. Jede Subroutine liest dabei nur die sie betreffenden Daten aus. Im Anschluss daran wird durch eine umgekehrte Anwendung der Mapping-Schritte ein Datenarray erzeugt, das die Daten in der von Magento benötigten Form abbildet. Diese werden dann wiederum über die Proxy-Klasse `Core_Mage_Api_Proxy` nach Magento übertragen. Als Rückgabe liefert jede Subroutine Angaben über die Anzahl der neu angelegten bzw. aktualisierten Datenobjekte. In der `finish` Methode wird lediglich überprüft, ob die Importdatei nach Abschluss der Operationen gelöscht, verschoben oder beibehalten werden soll.

4.6 Anbindungsmöglichkeiten für andere Warenwirtschaftssysteme

Die Möglichkeit, mit der Integrationslösung Mageport andere Warenwirtschaftsysteme an Magento anbinden zu können, ist eine der wichtigsten Anforderungen des Integrationsszenarios (vgl. Abschnitt 3.1, S. 60). Mageport soll die dafür benötigten Anpassungen leicht zugänglich und überschaubar halten. Damit wird die Integrationslösung für einen größeren Kreis von Nutzern interessant, die Magento Commerce mit einer Warenwirtschaft integrieren möchten. Es wird damit ein Schritt von einer Invididuallösung für eine bestimmte Branchensoftware hin zu einer Allgemeinlösung unternommen. Eine solche Allgemeinlösung hat – neben dem größeren potentiellen Anwenderkreis – den weiteren Vorteil, dass der Aufwand für einzelne Projekte bei wiederholter Verwendung der Integrationslösung, z.B. im Rahmen einer auf diesem Gebiet tätigen Firma,

immer geringer ausfällt, da die Integrationslösung selbst bereits fertig existiert und nur noch für den konkreten Integrationsfall konfiguriert bzw. angepasst werden muss.

Prinzipiell muss die Anbindung anderer Warenwirtschaftssysteme nach demselben Schema ablaufen: Die Daten der Warenwirtschaft kommen in einer XML-Austauschdatei auf das Mageportsystem, und in gleicher Weise stellt Mageport Daten für die Warenwirtschaft in Form einer XML-Austauschdatei bereit. Die Möglichkeit des Datenaustauschs über XML-Dateien stellt also die gemeinsame Grundlage dar, auf deren Basis Mageport mit verschiedenen Warenwirtschaftssystemen verwendet werden kann. Stellt die anzubindende Warenwirtschaft diese Möglichkeit nicht selbst zur Verfügung, so muss durch die Verwendung eines geeigneten Transformationswerkzeugs (vgl. Abschnitt 2.7.4, S. 55) eine Zwischentransformation vorgenommen werden, um aus der XML-Datei ein für die Warenwirtschaft verwendbares Datenformat zu erzeugen. Wenn sichergestellt ist, dass XML-Dateien für den Datenaustausch verwendet werden können, dann kann die Anbindung der Warenwirtschaft durch das Befolgen der folgenden Arbeitsschritte realisiert werden:

Analyse der XML-Dateistruktur

Zuerst sollte in der Warenwirtschaft eine beispielhafte XML-Datei erzeugt werden, die möglichst von allen von Mageport verwendeten Datenobjekten – Kunden, Bestellungen, Produkte und Katalogdaten – ein Beispielobjekt enthält. Diese Beispielobjekte sollten alle Datenattribute verwenden, die für die Integration zwischen Magento Commerce und der Warenwirtschaft von Relevanz sind. Diese Datei wird nun unter zwei Gesichtspunkten analysiert: Zum Einen wird die Struktur der Datei festgehalten. Das umfasst vor allem die Wurzelelemente unter denen die einzelnen Datenobjekte angeordnet sind, sowie die Verschachtelung der Attribute innerhalb der Datenobjekte. Zum Anderen muss überprüft werden, welche Werte die einzelnen Attribute beinhalten bzw. erfordern. Zu jedem zu verwendenden Attribut muss in Mageport in der Klasse `Core_Mapper_Mage` ein Attributschlüssel definiert sein, der wiederum mit einem Datenattribut aus Magento Commerce verknüpft ist.

Anlegen neuer Attributschlüssel

Mageport bringt die wichtigsten Attributschlüssel für jede Integrationsoperation bereits mit. Wenn sich bei der Analyse der XML-Dateistruktur dennoch herausstellt, dass zur Integration benöigte Attributschlüssel in Mageport

nicht vorhanden sind, dann müssen diese nun angelegt werden. Dies geschieht durch die Ergänzung des Attributschlüssel im entsprechenden Mapping-Array in der Klasse `Core_Mapper_Mage`. Es existieren zwei Arrays in der Klasse, nämlich `$soapMappings`, das bei Verwendung der SOAP API benutzt wird, und `$localMappings` für die Verwendung im Magento Appmode. Da im Magento Appmode prinzipiell die selben Klassen verwendet werden, die auch bei Verwendung der SOAP API angesprochen werden, sind die Rückgabearrays der beiden Schnittstellen weitgehend identisch. Damit sind zumindest derzeit auch die beiden Arrays identisch und das Array `$localMappings` wird durch Kopie des Arrays `$soapMappings` erzeugt. Es genügt daher die Ergänzung des neuen Attributschlüssels im Array `$soapMappings`. Für die weitere Beschreibung wird immer auf dieses Array Bezug genommen.

Das Array enthält zunächst vier Felder, die für die vier verschiedenen grundlegenden Datenobjekte stehen die die Integration berücksichtigt: Kunden-, Bestellungs-, Produkt- und Katalogdaten. Sie spiegeln die Grunddifferenzierung der verschiedenen Datenoperationen wieder. Der Aufbau von Mageport bedingt, dass bei jeder möglichen Einzeloperation (siehe die Abschnitte 3.8.3.1 und 3.8.3.2 ab Seite 97) immer nur ein Grundobjekt des Mappingarray verwendet wird. Die Daten und auch die Attributschlüssel können damit nicht vermischt angewendet werden. Jedes der vier Felder enthält nun wiederum ein Array, in dem die einzelnen Attributschlüssel für das jeweilige Datenobjekt aufgelistet und ihrer Entsprechung in den Ergebnisdaten aus Magento Commerce zugeordnet sind. Aus der Einschränkung, dass nur jeweils ein Feld für eine Integrationsoperation verwendet wird, folgt, dass es innerhalb der verschiedenen Felder identische Attributschlüssel geben kann. Diese müssen also nur innerhalb ihres Feldes eindeutig sein. So gibt es z.B. den Attributschlüssel `TITLE` sowohl bei den Produkt-, als auch bei den Katalogdaten. Er bezieht sich jedoch jeweils auf ein anderes Datenattribut eines anderen Datenobjekts: Bei den Katalogdaten bezeichnet er den Titel einer Kategorie, während er in den Produktdaten für die Bezeichnung eines Produkts steht.

Dem Attributschlüssel wird jeweils eine Zeichenfolge zugeordnet, die angibt, wo das jeweilige konkrete Datenattribut im Ergebnisarray aus Magento Commerce zu finden ist. Der Schrägstrich (`/`) bedeutet dabei „gehe eine Ebene tiefer“ und ermöglicht es so, auch im Ergebnisarray verschachtelte Attribute auszulesen. Das Rautezeichen (`#`) bedeutet „numerische Ebene“ und repräsentiert eine Verschachtelungsebene im Ergebnisarray, in der eine Vielzahl von strukturell gleichen Unterarrays in einem sammelnden Array vereint sind. Es repräsentiert also ein Container-Element. Die Abbildung 4.4 verdeutlicht die Wirkungsweise des Schrägstrichs und des Rautezeichens in einem Ergebnisar-

```
public $soapMappings = array(

    Core_Xml_Grouptags::DATAOBJECT_ORDERS => array(
        'STATUS'           => 'status',
        'CURRENCY'         => 'order_currency_code',
        'CUSTOMER/ID'      => 'customer_id',
        'CUSTOMER/EMAIL' => 'customer_email',
 [...]
        'ARTICLES/#/SKU'           => 'items/#/sku',
        'ARTICLES/#/TITLE'         => 'items/#/name',
        'ARTICLES/#/ENTITYPRICE' => 'items/#/base_price',
        'ARTICLES/#/QUANTITY'      => 'items/#/qty_ordered',
 [...]
        'SHIPPINGADDRESS/FIRSTNAME'  => 'shipping_address/firstname',
        'SHIPPINGADDRESS/MIDDLENAME' => 'shipping_address/middlename',
 [...]
        'BILLINGADDRESS/FIRSTNAME'  => 'billing_address/firstname',
        'BILLINGADDRESS/MIDDLENAME' => 'billing_address/middlename',
 [...]
    ),

    Core_Xml_Grouptags::DATAOBJECT_CUSTOMERS => array(
        'CUSTOMER_ID' => 'customer_id',
        'UPDATED'       => 'updated_at',
        'FIRSTNAME'     => 'firstname',
        'EMAIL'         => 'email',
 [...]
        'ADDRESS/#/DEFAULT_BILLING' => 'addresses/#/is_default_billing',
        'ADDRESS/#/FIRSTNAME'       => 'addresses/#/firstname',
        'ADDRESS/#/POSTCODE'        => 'addresses/#/postcode',
        'ADDRESS/#/CITY'            => 'addresses/#/city',
 [...]
    ),

    Core_Xml_Grouptags::DATAOBJECT_PRODUCTS => array(
        'PRODUCT_ID' => 'product_id',
        'SKU'          => 'sku',
        'TITLE'        => 'name',
        'PRICE'        => 'price',
 [...]
    ),

    Core_Xml_Grouptags::DATAOBJECT_CATALOG => array(
        'CATEGORY_ID' => 'category_id',
        'TITLE'         => 'name',
        'LEVEL'         => 'level',
        'POSITION'      => 'position',
        'PARENT_ID'     => 'parent_id',
 [...]
    )
);
```

Tabelle 4.5: Teil des Soapmapping-Array von Mageport

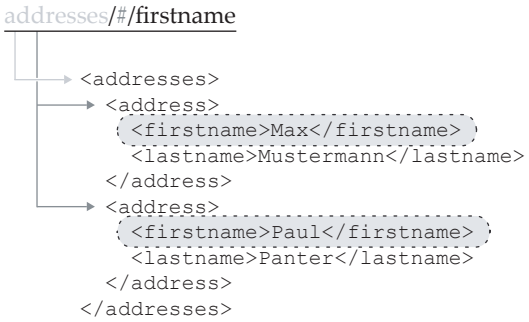

Abbildung 4.4: Wirkungsweise der Mapping-Sonderzeichen

ray. Beide Zeichen sind mit gleicher Wirkung auch für die Attributschlüssel anwendbar.

Der schwierigste Teil beim Ergänzen von neuen Attributschlüsseln ist jedoch, das entsprechende Feld im Ergebnisarray von Magento Commerce zu lokalisieren. Dafür bietet die Konfigurationsdatei in Mageport die Option `<settings>` → `<dumpValues>`. Wird diese zusammen mit dem Debug-Modus aktiviert, so wird beim Aufruf einer Export-Operation das Ergebnisarray aus Magento per PHP `var_dump()` ausgegeben. Dann kann anhand des Outputs das benötigte Datenfeld lokalisiert und in der beschriebenen Notation einem Attributschlüssel zugeordnet werden. Anschließend muss die Ausgabe wieder aus dem Quellcode entfernt werden.

Erstellen der Mapping-XML-Datei

Wenn alle benötigten Attributschlüssel vorhanden bzw. angelegt sind, kann die eigentliche Mapping-XML-Datei erstellt werden. Die Datei muss im Verzeichnis `Mappings` angelegt werden und die Dateiendung `.xml` aufweisen. Der Inhalt der Datei setzt sich zusammen aus der Struktur der analysierten XML-Austauschdatei der Warenwirtschaft, und den anstelle der konkreten Attributwerte eingesetzten Attributschlüsseln. Für jedes Datenobjekt wird also ein abstraktes Element erzeugt, dass alle Attribute als Tags oder Verschachtelungen von Tags enthält, die für die Integration relevant sind. Als Werte für diese Attributtags sind aber statt der konkreten Werte die Attributschlüssel von Mageport enthalten. Diese werden dann während der Integrationsoperation durch die konkreten Werte ersetzt.

Wichtig ist außerdem, darauf zu achten, dass jedes Datenelement von einem weiteren Element umschlossen wird, welches später als Container für mehrere

```
<wawiExchangeDocument>
  <onlineSales>
    <orders>
      <order>
        [...]
      </order>
    </orders>
    <customers>
      <customer>
        [...]
      </customer>
    </customers>
    <products>
      <product>
        [...]
      </product>
    </products>
    <catalog>
      <product>
        [...]
      </product>
    </catalog>
  </onlineSales>
</wawiExchangeDocument>
```

Abbildung 4.5: Group Tags in der Mapping-Datei von Mageport

gleichartige Datenelemente dient. Wenn also z.B. alle exportierten Bestellungen in der Datei in je einem Tag namens `<order>` gemapped werden sollen, so muss das Tag `<order>` in der Mappingdatei (das alle relevanten Datenattribute als Tags und die Attributschlüssel als Tagwerte enthält) von einem weiteren Tag umschlossen werden – z.B. `<orders>` genannt. Diese Container-Elemente werden in Mageport als Gruppentags (Group Tags) bezeichnet.

Eintragen der Gruppentags

Die Gruppentags müssen Mageport bekannt sein. Sie sind die einzigen Anhaltspunkte, nach denen die Anwendung die Mapping-XML-Datei durchsucht, um die Mappings für ein bestimmtes Datenobjekt zu finden. Deshalb müssen sie nach der Erstellung der XML-Datei in Mageport gespeichert werden. Dazu muss in der Klasse `Core_Xml_Grouptags` die Variable `$groupTags` entsprechend angepasst werden, sodass die Werte den Namen der jeweiligen Gruppentags aus der Mappingdatei entsprechen.

Ändern der Konfigurationsdatei

Der letzte Schritt besteht darin, die Konfigurationsdatei `Config.xml` anzupassen. Zunächst muss hier eingestellt werden, dass Mageport die neu erstellte

Mappingdatei verwendet. Dazu wird der Wert von `<settings>` → `<erp>` auf den Namen der neu erstellten Mappingdatei – ohne die Dateiendung `.xml` – gesetzt. Damit verwendet Mageport die neu erstellte Mappingdatei als Grundlage für die Integrationsoperationen. Weiterhin sollten noch die Einstellungen unter `<xml>` überprüft werden, speziell die Einstellungen unter`<xml>` → `<import>` → `<overwrite>`, die das Verhalten von Mageport bei Importoperationen wesentlich mitbestimmen. Zu den genauen Auswirkungen und Einstellmöglichkeiten der Konfigurationsdatei siehe Anhang A.2, Seite XX.

5 Fazit

5.1 Zusammenfassung

Die Integration von Einzelanwendungen aus verschiedenen Unternehmensbereichen zu integrierten Gesamtsystemen ist eine immer wichtiger werdende Aufgabe bei der Realisierung von modernen und effektiven Unternehmensstrukturen. Voraussetzung für eine fundierte Abwägung von Chancen und Risiken, für die Kalkulation von Aufwand und Nutzen und die Entscheidung für das richtige Werkzeug ist fachliches Grundwissen über die Problemdimensionen der Integration, über bekannte Integrationsansätze und -methoden. Die vorliegende Arbeit hat zu diesem Zweck eine zusammenfassende Einführung in dieses weitverzweigte Feld vorgelegt. Der aktuelle Stand der grundlegenden theoretischen Erkenntnisse zur Integration war hier ebenso Thema, wie ein Überblick über in der Paxis verbreitete Integrationstechniken. In besonderer Ausführlichkeit wurde auf das Thema Web Services eingegangen, da diese Technologie zur Zeit als Mittel der Wahl für viele Integrationsvorgänge angesehen wird.

Integrationsvorhaben sind auch für den Geschäftsbereich des E-Commerce relevant. Gerade hier sind durch Integrationslösungen mit den unternehmensinternen Planungs- und Managementwerkzeugen deutliche Verbesserungen hinsichtlich Effizienz, Kompetenz, Kundenorientierung und Flexibilität der Unternehmen erreichbar. Im Zentrum des E-Commerce muss jedoch immer der Kunde und das Ermöglichen eines optimalen Einkaufserlebnis für ihn stehen. Das neue E-Commerce-System Magento Commerce setzt in dieser Hinsicht hohe Maßstäbe und wird sich auch aus diesem Grund wohl zu einem der bedeutendsten Systeme im OpenSource-Bereich entwickeln. Durch seine konsequente und klare Struktur bietet es auch für Integrationsoperationen gute Voraussetzungen und bringt zwei wesentliche Schnittstellen hierfür mit, die Magento Core Api und den Appmode. Eine prototypische Integrationslösung zur Anbindung an Warenwirtschafts- und Prozessmanagmentsysteme in Unternehmen mit diesen Schnittstellen konnte im Rahmen der Arbeit realisiert werden. Auch wenn das System in der vorgestellten Form noch nicht ausgereift genug für einen realen Einsatz ist, konnte die prinzipielle Eignung und Struktur einer solchen Anbindung gezeigt werden.

Die Eignung verschiedener Schnittstellen, sowohl des E-Commerce-System als auch des unternehmensinternen Branchensystem, wurden untersucht, auf ihre Funktion und Eignung geprüft und in der Integrationslösung praktisch verwendet. Insbesondere der Magento Appmode wurde als leistungsfähiger, vielversprechender Ansatz für die integrationsorientierte Kommunikation mit Magento ausgemacht und dargestellt. Außerdem wurden Funktionalitäten geschaffen

und dargestellt, die es ermöglichen, die Integrationslösung flexibel mit verschiedenen Warenwirtschaftssystemen zu verbinden und wiederzuverwenden.

5.2 Leistungsfähigkeit der Integrationslösung

Im Rahmen der Arbeit wurde für die Realisierung der Integrationslösung die Software Mageport entwickelt. Momentan ist diese noch ein Prototyp, der belegt, dass die Nutzung der verfügbaren Schnittstellen für eine Integrationslösung in der beschriebenen Weise möglich ist, zum gewünschten Ergebnis führt und in der Praxis denkbar ist. Es konnten alle eingangs definierten Integrationsoperationen in Mageport umgesetzt werden. Die Eignung der verwendeten Schnittstellen konnte damit nachgewiesen werden. Aufgrund des modularen Aufbaus von Mageport ist auch die Ergänzung weiterer Schnittstellen, sowie weiterer Aufrufe der bereits implementierten Schnittstellen möglich. Zusätzlich wurden verschiedene Benachrichtigungsfunktionen implementiert, die sich aus den Anforderungen des Integrationsszenarios für einen realen Betrieb definierten. Die Anbindungsmöglichkeiten für andere Warenwirtschaftssysteme als das in dieser Arbeit verwendete ALEA Commerce Suite ist ebenfalls ein Leistungsmerkmal der Integrationslösung, das in der Arbeit gefordert wurde und entsprechend umgesetzt werden konnte.

Die Verwendung der zusätzlichen Komponenten `crontab` und `rsync` für Teilaufgaben der Integrationslösung macht diese sehr flexibel einsetzbar und auch flexibel einstellbar. Die Intervalle der Aufrufe der Integrationsoperationen sind so völlig individuell konfigurierbar. Die gute Verfügbarkeit und Erprobtheit der Komponenten lässt Fehler oder Nachteile aufgrund ihrer Verwendung nicht erwarten.

Mit der Integrationslösung konnte ein hoher Grad der Automatisierung erreicht werden. Der Austausch der wesentlichen Datenobjekte mit der höchsten Relevanz in Online-Shops – Kundendaten, Produktdaten, Katalogdaten und Bestellungen – konnte in beide Integrationsrichtungen automatisiert werden. Eine vollständige Automatisierung des Online-Shops konnte nicht verwirklicht werden. Nötig zu ergänzen sind vor allem Daten zur Produktlagerhaltung, sowie zur Produktpräsentation gehörige Medien, v.a. Produktabbildungen. Für die Steuerung anderer Funktionen des Online-Shops Magento Commerce, z.B. Marketing-Tools oder Newsletter gibt es derzeit keine Schnittstellen zur Realisierung einer Automatisierung.

5.3 Ansätze für Verbesserungen

Für den Einsatz der Integrationslösung in realen Unternehmen sind weitere Arbeiten und Verbesserungen nötig. In erster Linie müssen systematische Tests mit umfangreichen Daten durchgeführt werden, die zeigen, ob sich die Integrationslösung unter der Beanspruchung durch große Datenmengen den Anforderungen entsprechend verhält.

Nötig ist weiterhin auch die Verarbeitung von Lagerhaltungsdaten zu den Produkten im Online-Shop, um Aussagen zu deren Verfügbarkeit und Beschaffung besser in die Integrationsmaßnahmen zu integrieren. Diese sind für einen realen Betrieb unverzichtbar. Durch eine Erweiterung der bereits vorhandenen Schnittstellen-Aufrufe sollte das problemlos möglich sein. Die Verarbeitung von Produktabbildungen durch die Integrationsschnittstelle ist ein weiterer wichtiger Punkt, der einer Lösung bedarf. Auch hier existieren prinzipiell Schnittstellenmethoden in Magento Commerce.

Die Nutzung der XML-Dateien zum Austausch der Daten mit der Warenwirtschaft ist aufwändig und hat einen relativ großen Anteil an der Laufzeit und dem Speicherverbrauch der Integrationslösung. Hier wäre zu prüfen, welche Optimierungen in der Integrationslösung vorgenommen werden können, bzw. welche anderen Formate oder Schnittstellen zu der Warenwirtschaft außerdem noch verwendet werden können.

Einer Klärung bedarf auch die Frage der Verfügbarmachung der Integrationslösung. Hier ist eine Veröffentlichung als OpenSource-Software für die Magento-Community bei Erreichen eines angemessenen und praxistauglichen Entwickungsstands angedacht.

A Anhang

A.1 Datenattribute der Integrationslösung

Die folgenden Tabellen listen alle Datenattribute auf, die in der Integrationslösung im Rahmen der Arbeit berücksichtigt wurden. Sie sind nach den Grundobjekten der Integrationsoperationen unterteilt. Die erste Spalte enthält jeweils die Bezeichnung bzw. Bedeutung des Attributes im Klartext. Die zweite Spalte führt den internen Attributbezeichner der Integrationslösung auf. Die dritte Spalte enthält den Feldnamen im Ergebnis-Array einer entsprechenden Anfrage an Magento. Die letzte Spalte enthält die Bezeichung des entsprechenden Tags in der XML-Austauschdatei für die ALEA Commerce Suite .

Ein oder mehrere Pfeile (→) zeigen an, dass sich das jeweilige Attribut in einer Verschachtelung unter einem anderen Attribut befindet. Ist ein Attribut selbst ein Datenobjekt bzw. -unterobjekt, so wird es in der Tabelle **fett** hervorgehoben.

Katalogdaten

Attribut	Schlüssel Mageport	Feldname Magento	Feldname ALEA
Kategorie	–	–	–
→ Id	CATEGORY_ID	category_id	id
→ Titel	TITLE	title	title
→ Kategorie aktiv	ENABLED	is_active	enabled
→ Kategorieebene	LEVEL	level	level
→ Position in der Kategorie	POSITION	position	position
→ Id der Elternkategorie	PARENT_ID	parent_id	parent_id
→ URL-Key	URLKEY	url_key	urlkey
→ URL-Pfad	URLPATH	url_path	urlpath
→ Ids der Kindkategorien	CHILDREN/#/VALUE	children/#	children → child → id

Tabelle A.1: Datenattribute für Katalogdaten

Kundendaten

Attribut	Schlüssel Mageport	Feldname Magento	Feldname ALEA
Kunde	–	–	**customer**
→ Kunden-ID	CUSTOMER_ID	customer_id	→ id
→ zuletzt geändert	UPDATED	updated_at	→ updated
→ Vorname	FIRSTNAME	firstname	→ name *(multi)*
→ Mittelname	MIDDLENAME	middlename	→ name *(multi)*
→ Nachname	LASTNAME	lastname	→ name *(multi)*
→ Emailadresse	EMAIL	email	→ email
→ **Adresse**	–	**address**	–
→ → Standardrechnungsadresse	DEFAULT_BILLING	→ is_default_billing	→ default_billing
→ → Standardversandadresse	DEFAULT_SHIPPING	→ is_default_shipping	→ default_shipping
→ → Straße mit Hausnr.	STREET	→ street	→ street
→ → Postleitzahl	POSTCODE	→ postcode	→ postcode
→ → Stadt/Ort	CITY	→ city	→ city
→ → Land (ID-Kennzahl)	COUNTY_ID	→ country_id	→ country
→ → Telefon	TELEPHONE	telephone	→ telephone
→ → Telefax	TELEFAX	telefax	→ telefax
→ → Region	REGION	→ region	→ region
→ → Region Kennzahl	REGION_ID	→ region_id	→ region_id

Tabelle A.2: Datenattribute für Kundendaten

Produktdaten

Attribut	Schlüssel Mageport	Feldname Magento	Feldname ALEA
Produkt	–	–	–
→ Id	PRODUCT_ID	product_id	id
→ SKU	SKU	sku	sku
→ Typ	TYPE	type	type
→ Status (Kennzahl)	STATUS_ID	status	status
→ Attributset	ATTRIBUTE_SET	set	attributeSet
→ Erstellungsdatum	CREATED_AT	created_at	created
→ Änderungsdatum	UPDATED_AT	updated_at	updated
→ Name	TITLE	name	name
→ Beschreibung	DESCRIPTION	description	description
→ Kurzbeschreibung	SHORTDESCRIPTION	short_description	short_description
→ Preis	PRICE	price	price
→ Steuerklasse (Kennzahl)	TAXCLASS	tax_class_id	tax_class_id
→ URL-Key für Navigation	URLKEY	url_key	urlkey
→ URL-Pfad	URLPATH	url_path	urlpath
→ Metatitel	METATITLE	meta_title	metatitle
→ Metakeywords	METAKEYWORDS	meta_keywords	metakeywords
→ Metadescription	METADESCRIPTION	meta_description	metadescription
→ Website - IDs	WEBSITE/#/VALUE	website/#	websites → website → id
→ Kategorien - IDs	CATEGORY/#/VALUE	categories/#	categories → category → id

Tabelle A.3: Datenattribute für Produktdaten

Bestellungsdaten

siehe bitte umseitig

Attribut	Schlüssel Mageport	Feldname Magento	Feldname ALEA
Bestellung	–	–	–
→ Bestellzeitpunkt	DATETIME	created_at	datetime
→ Status	STATUS	status	status
→ Währung	CURRENCY	order_currency_code	currency
→ Versandart	SHIPPINGMETHOD	shipping_method	shippingMethod
→ Versandkosten	SHIPPINGAMOUNT	shipping_amount	shippingAmount
→ Rechnungspreis ohne Versand	SUBTOTAL	subtotal	subtotal
→ Rechnungspreis inkl. Versand	GRANDTOTAL	grand_total	grand_total
→ **Kunde**	**CUSTOMER**	–	**customer**
→ → ID	→ ID	customer_id	→ id
→ → Vorname	→ FIRSTNAME	customer_firstname	→ firstname
→ → Mittelname	→ MIDDLENAME	customer_middlename	→ middlename
→ → Nachname	→ LASTNAME	customer_lastname	→ lastname
→ → Email	→ EMAIL	customer_email	→ email
→ **Lieferadresse**	**SHIPPINGADDRESS**	**shipping_address**	**shippingAddress**
→ → Vorname	→ FIRSTNAME	→ firstname	→ firstname
→ → Mittelname	→ MIDDLENAME	→ middlename	→ middlename
→ → Nachname	→ LASTNAME	→ lastname	→ lastname
→ → Firmenname	→ COMPANY	→ company	→ company
→ → Straße + Hausnr.	→ STREET	→ street	→ street
→ → Stadt/Ort	→ CITY	→ city	→ city
→ → Postleitzahl	→ ZIPCODE	→ postcode	→ zipcode
→ → Land (ID-Kennzahl	→ COUNTRY	→ country_id	→ country
→ → Telefon	→ TELEPHONE	→ telephone	→ telephone
→ **Rechnungsadresse**	**BILLINGADDRESS**	**billing_address**	**billingAddress**
→ → Vorname	→ FIRSTNAME	→ firstname	→ firstname
→ → Mittelname	→ MIDDLENAME	→ middlename	→ middlename
→ → Nachname	→ LASTNAME	→ lastname	→ lastname
→ → Firmenname	→ COMPANY	→ company	→ company
→ → Straße + Hausnr.	→ STREET	→ street	→ street
→ → Stadt/Ort	→ CITY	→ city	→ city
→ → Postleitzahl	→ ZIPCODE	→ postcode	→ zipcode
→ → Land (ID-Kennzahl	→ COUNTRY	→ country_id	→ country
→ **Zahlungsweise**	**PAYMENT**	**payment**	**payment**
→ → Zahlungsart	→ METHOD	→ method	→ method
→ → Zahlungsart ID	→ METHOD_ID	→ payment_id	→ methodId
→ → **Kreditkarte**	→ **CREDITCARD**	–	→ **creditcard**
→ → → Nummer (verschlüsselt)	→ → NUMBERENC	→ cc_number_enc	→ → numberenc
→ → → letzte 4 Stellen d. Nummer	→ → NUMBERLAST4	→ cc_last4	→ → last4
→ → → gültig bis Monat	→ → EXPMONTH	→ cc_exp_month	→ → validTo *(multi)*
→ → → gültig bis Jahr	→ → EXPYEAR	→ cc_exp_year	→ → validTo *(multi)*
→ → → Eigentümer	→ → OWNER	→ cc_owner	→ → owner
→ **Artikel**	**ARTICLES**	**articles**	**items**
→ → SKU	→ SKU	→ sku	→ id
→ → Name	→ NAME	→ name	→ title
→ → Grundpreis	→ ENTITY_PRICE	→ base_price	→ entityPrice
→ → Originalpreis	→ ORIGINAL_PRICE	→ original_price	→ entityPrice
→ → Postenpreis	→ ITEMTOTAL	→ row_total	→ itemTotal
→ → Menge	→ QUANTITY	→ qty_ordered	→ quantity
→ → Steuer (Prozent)	→ TAXPERCENT	→ tax_percent	→ taxpercent
→ → Steuer (Betrag)	→ TAXAMOUNT	→ tax_amount	→ taxamount
→ → Ermäßigung	→ DISCOUNT	→ discount_amount	→ discount

Tabelle A.4: Datenattribute für Bestellungsdaten

A.2 Konfigurationslemente der Config.default.xml

settings → debug
Schaltet den Entwicklermodus an oder aus
Standardwert: *false*

settings → dumpValues
Aktiviert im Entwicklermodus bei Exportfunktionen die Ausgabe der Magento Datenarrays
Standardwert: *false*

settings → useapi
Zu benutzende Schnittstelle zu Magento. Mögliche Werte: `appmode` und `soap`
Standardwert: *appmode*

settings → erp
Zu benutzende Mappingdatei, ohne Endung `.xml`
Standardwert: *Alea*

settings → saveExecutionTime
Aktiviert die Speicherung der letzten Ausführungszeit einer Operation.
Standardwert: *true*

settings → useExecutionTime
Aktiviert die Verwendung der letzten Ausführungszeit der Operation als Wert für den Parameter `-since`, wenn dieser nicht angegeben wurde.
Standardwert: *true*

xml → export → path
Ordner, in dem exportierte XML-Dateien gespeichert werden
Standardwert: */server/Mageport/Data/Export/*

xml → export → filename
Dateinamensmuster für XML-Exportdateien, ohne Endung `.xml`
Standardwert: *export*

xml → export → overwrite
XML-Exportdateien mit gleichen Namen überschreiben?
Standardwert: *false*

xml → import → path
Ordner, in dem zu importierende XML-Dateien gesucht werden
Standardwert: */server/Mageport/Data/Import/*

xml → import → filename
Dateinamensmuster für XML-Importdateien, ohne Endung `.xml`
Standardwert: *import*

xml → import → onSuccess
Was passiert nach dem Import mit der XML-Datei?
Mögliche Werte: `move` (verschieben), `delete` (löschen) oder leer lassen für '"keine Aktion'"

xml → import → overwrite → customerData
Kundendaten beim Import aktualisieren?
Standardwert: *false*

xml → import → overwrite → customerAddresses
Kundenadressen beim Import aktualisieren?
Standardwert: *false*

xml → import → overwrite → catalogCategories
Katalog- und Kategoriedaten beim Import überschreiben?
Standardwert: *true*

xml → import → overwrite → productData
Produktdaten beim Import überschreiben?
Standardwert: *true*

exceptionLogging → email
Fehlerberichte per Email senden
Standardwert: *true*

exceptionLogging → emailAddress
Emailadresse für Fehlerbenachrichtigung

exceptionLogging → logfile
Fehlerberichte in Logdatei speichern
Standardwert: *true*

exceptionLogging → logfilePath
Ordner, in dem die Fehlerberichte gespeichert werden
Standardwert: */server/Mageport/Logs/*

messageLogging → enabled
Sollen Statusmitteilungen in Logdatei gespeichert werden?
Standardwert: *true*

messageLogging → logfilePath
Ordner, in dem die Statusmitteilungen gespeichert werden
Standardwert: */server/Mageport/Logs/*

smtp → replyAddress
Emailadresse, die als Absender für Benachrichtiungen verwendet wird

smtp → noreplyAddress
Emailadresse, die als Absender für Benachrichtiungen verwendet wird

smtp → host
Mailserver des zu verwendenden Mailaccounts

smtp → username
Nutzername des zu verwendenden Mailaccounts

smtp → password
Passwort des zu verwendenden Mailaccounts

magento → basePath
Basisordner der lokalen Magento-Installation (nur für Appmode benötigt)
Standardwert: */var/www/Magento/*

magento → api → soap → url
URL-Teil unter dem die WSDL-Beschreibung der SOAP API abrufbar ist, relativ zur Startseite angegeben
Standardwert: *api/soap/?wsdl*

magento → api → soap → username
Nutzername des API Benutzers

magento → api → soap → apikey
Nutzername des API Benutzers

A.3 Datenträger mit der Integrationslösung „Mageport“

[Eingeklebter Datenträger mit der Integrationslösung „Mageport“]
– Nicht enthalten in Bibliotheksexemplaren –

Glossar

Adressraum Bereich im Hauptspeicher eines Rechners, in dem eine definierte Anzahl von Objekten für den Prozessor vorgehalten werden kann. [Heinrich u. a., 2004, 699].

API (Application Programming Interface) Schnittstelle mit festgelegten Befehlen und Argumenten, mit denen ein Anwendungssystem durch ein anderes System zur Ausführung von Operationen und ggf. zur Rückgabe von Daten veranlasst werden kann. [Heinrich u. a., 2004, 65].

Caching Zwischenspeichern von einmal erzeugten Inhalten in einem Puffer. Werden diese Inhalte erneut angefragt, werden sie aus dem Puffer geladen und nicht erneut erzeugt. Das reduziert den Aufwand und beschleunigt die Antwort auf die Anfrage. Andererseits geben die gecachten Inhalte u.U. nicht die aktuellen Daten wieder, sondern nur die Daten, die zum Zeitpunkt des Cachings aktuell waren.

CGI-Binary Eine Binärdatei, die auf einem Server eine Schnittstelle zur Verfügung stellt, über die externe Anwendungen (Browser) Skripte auf dem Server aufrufen können. Die Ergebnisse des Skriptes werden an die Anwendung zurück gesendet und in der Regel einer grafischen Ausgabe zugeführt (HTML). Als Übertragungsprotokoll wird → HTTP verwendet. CGI steht für Common Gateway Interface. [Schneider / Werner, 2007, 403f].

CLI-Binary Eine Binärdatei, die es einem Serverdienst erlaubt, aufgerufene Skripte loakel auf der Kommandozeile (Console) des Betriebssystems auszuführen und die Ausgaben wiederum der Kommandozeile, anstatt einer grafischen Ausgabe zuzuführen. CLI steht für Command Line Interface.

CSV (Comma Separated Values) ein Dateiformat, das sich besonders für den Austausch von in Tabellen strukturierten Daten zwischen Anwendungen eignet. Jede Zeile stellt eine Tabellenzeile dar, die einzelnen Spalten werden durch ein festgelegtes Trennzeichen (meist ; oder ,) getrennt. Aufgrund seiner einfachen Struktur ist es im Bereich der Internetanwendungen recht weit verbreitet und wird von vielen Anwendungen unterstützt. Aber auch Officeprogramme wie Excel können CSV-Dateien verarbeiten.

DCE (Distributed Computing Environment) verteilte Rechenumgebung, Verfahren, das die physische und logische Verteilung von Rechnern und Adress-

räumen weitgehend überbrückt und Anwendungen so zusammenarbeiten lässt, als befänden sie sich im selben Adressraum.

EAI (Enterprise Application Integration) Oberbegriff für die Integration von Anwendungen und Daten in einem Unternehmen oder zwischen Unternehmen zur Verbesserung der Informationsinfrastruktur. [Heinrich u. a., 2004, 210].

ESB (Enterprise Service Bus) standardisierter Bus zur nachrichtenbasierten Integration von Anwendungen oder Anwendungskomponenten.

ETL-Tools Extract, Transform, Load; bezeichnet Werkzeuge, die Daten aus bestehenden Datenquellen extrahieren, diese in ein neues Datenschema transformieren und schließlich in eine neue Datenquelle laden.

XML (Extensible Markup Language) die heutzutage meistgenutzte Meta-Auszeichnungspsrache. Sie definiert einen Rahmen für eine Vielzahl von anwendungsspezifischen Auszeichnungsformaten und gibt gleichzeitig Regeln vor, wie Parser die Dokumente auf Gültigkeit prüfen und Daten aus den Dokumenten auslesen können. [Rechenberg / Pomberger, 2006, 289].

XMLRPC (Extensible Markup Language Remote Procedure Call) Ein Netzwerkprotokoll, das sowohl für den asynchronen Austausch von Daten, als auch für →RPCs geeignet ist. Als Austauschformat für die Daten wird XML verwendet.

Framework Wörtl. übers. *Rahmenwerk*; Sammlung von Klassen, die Lösungen für wiederkehrende Problemstellungen bei der Entwicklung von Anwendungen zur Wiederverwendung zur Verfügung stellt [Rechenberg / Pomberger, 2006, 816]. Frameworks gibt es für die verschiedensten Anwendungsarten und Programmiersprachen.

PHP (Hypertext Preprocessor) eine weitverbreitete interpretierte Sprache, die speziell für die Entwicklung von Internetanwendungen geeignet ist. Offizielle Website: http://php.net.

HTTP (Hypertext Transfer Protocol) Statusloses Protokoll für die Datenübertragung im Internet. Es dient der Adressierung von Ressourcen über →Uniform Resource Locators und wickelt Interaktionen von Client und Server ab. Es ist statuslos, d.h. jede Anfrage eines Clients stellt eine neue Verbindung dar, eine Speicherung von Objekten über die Verbindung hinaus findet nicht statt.

IP (Internet Protocol) Verbindungsloses paketorientiertes Protokoll zum Transport von Datenpaketen über mehrere Netzknoten hinweg. Die Datenpakete werden unahängig voneinander gesendet, es besteht keine Gewährleistung der Einhaltung einer bestimmten Reihenfolge oder der Ablieferung beim Empfänger. [Lipinski, 2001, 257].

Interpretieren Vorgang der Umwandlung von Anweisungen einer menschenverständlichen Quellsprache in eine Maschinensprache durch einen Interpreter mit sofort anschließender Ausführung des Ergebnis. Vgl. → Kompilieren.

Intranet Unternehmensinternes Netzwerk, das wie das Internet auf TCP/IP aufbaut, aber nicht öffentlich erreichbar ist. Über spezielle Dienste (VPN) können Mitarbeiter auch von außerhalb auf ein Intranet zugreifen. Diese Ausprägung wird auch als Extranet bezeichnet.

Kompilieren Vorgang der Umwandlung von Anweisungen einer menschenverständlichen Quellsprache in eine Maschinensprache durch einen Compiler. Dabei wird das Quellprogramm auf die Einhaltung der Systax der Maschinensprache geprüft. Im Unterschied zum Vorgang des → Interpretieren wird das Ergebnis nicht sofort ausgeführt, sondern nur ein Ergebnis in der Maschinensprache erstellt, das später ausgeführt werden kann.

OSI Schichtenmodell Durch die Internationale Standardisierungs-Organisation (ISO) entworfenes Referenzmodell zur Steuerung der Kommunikationsvorgänge zwischen Anwendungssystemen. Jede Schicht besitzt einen eigenen Aufgabenbereich, verwendet eigene Kommunikationsprotokolle, greift auf Daten tieferliegender Schichten zu und gibt Daten an darüberliegende Schichten weiter.

Parser Programm zur syntaktischen Analyse eines anderen Programms [Heinrich u. a., 2004, 487].

PEAR Codingstandards eine von den → PEAR-Entwicklern verfasste umfangreiche Liste von Vorgaben für → PHP-Programmierer, die den erzeugten Code durch standardisierte Namens-, Einrückungs-, Formatierungs- und Dokumentationskonventionen leichter lesbar und besser wartungfähig macht. Vor allem bei Projekten mit mehreren Programmierern wird dadurch der Quellcode übersichtlich, einheitlich und verständlich gehalten.

PEAR (PHP Extension and Application Repository) ein Framework, das über ein eigenes Verteilungssystem wiederverwendbare PHP-Komponenten für verschiedenste Zwecke bereitstellt. Es ist sehr modular aufgebaut, jeder Benutzer entscheidet selbst, welche Komponenten er installieren möchte.

RPC (Remote Procedure Call) Entfernter Funktionsaufruf auf einem anderen physikalischen System über eine dafür vorgesehene Schnittstelle. Die Funktion wird auf dem entfernten System abgearbeitet und das Ergebnis an das aufrufende System zurück gesendet. Die Kommunikation läuft meist synchron ab, d.h. das aufrufende System wird solange blockiert, bis eine Antwort eintrifft.

Screen Scraping wörtl. übers. Abkratzen des Bildschirms, Aufbereitung eines zur graphischen Anzeige bestimmten Datenstroms für andere Zwecke als die

Anzeige. Wird häufig verwendet, um Daten aus Benutzungsschnittstellen für die automatische Weiterverarbeitung zu extrahieren.

SSH (Secure Shell) Im engeren Sinne ein Netzwerkprotokoll um eine verschlüsselte Verbindung zu einem entfernten Rechner aufzubauen. In der Praxis werden auch die in Client/Server-Architektur aufgebauten Programme zur Verwendung des SSH-Protokolls so bezeichnet. In verschiedenen Implementierungen liegt SSH für alle gängigen Systemplattformen (Unix-Derivate, Windows, IBM) vor.

Skalierbarkeit Fähigkeit einer Software mit den Anforderungen zu wachsen. Eine gut skalierbare Anwendung kann bei einer Verdopplung der Systemressourcen doppelt soviel leisten – z.B. doppelt soviele Aufgaben in einer vorgegebenen Zeit durchführen – und diese Skalierung wiederholen. Eine schlecht skalierbare Anwendung benötigt für eine Verdoppelung der Leistung bspw. eine Verfünffachung der Ressourcen, oder kann im schlimmsten Fall eine solche Verdoppelung gar nicht realisieren.

SOAP Ein Netzwerkprotokoll, das sowohl für den asynchronen Austausch von Daten, als auch für → RPCs geeignet ist. SOAP ist eine Weiterentwicklung des → XMLRPC.

SCM (Supply Chain Management) Instrument zur ganzheitlichen, unternehmensübergreifenden Gestaltung von Lieferketten [Ralev, 2004, 3].

TCP (Transmission Control Protocol) Ein verbindungsorientiertes Transportprotokoll für die paketbasierte Übertragung auf Schicht 3 des OSI-Modells. TCP baut auf dem → Internet Protocol (IP) auf und stellt vor der Übertragung eine gesicherte Verbindung zwischen den beteiligten Knoten her. Die wesentlichen Aufgaben des TCP sind das Verbindungsmanagement, die Flusskontrolle und die Fehlerbehandlung. [Lipinski, 2001, 449].

UML (Unified Modelling Language) Von der Object Management Group (OMG), einem internationalen Konsortium zur Entwicklung von Standards für die objektorientierte Programmierung, entwickelte Modellierungssprache für Softwaresysteme. UML kann sowohl für die Modellierung von Datenschemata, als auch zur Modellierung von Programmabläufen, Programmarchitekturen und aller weiterer Aspekte der Softwaremodellierung eingesetzt werden.
Offizielle UML-Projekthomepage: http://www.uml.org.

URL (Uniform Resource Locator) Form der Adressierung von Rechnern im Internet mit eindeutigen Namen.

Usability das Ausmaß, in dem ein Produkt durch bestimmte Benutzer in einem bestimmten Nutzungskontext genutzt werden kann, um bestimmte Ziele effektiv, effizient und zufriedenstellend zu erfüllen [Schulz, 2009, 58].

VM (Virtual Machine) Bezeichung für eine Software, die auf einem physischen Rechner einen oder mehrere virtuelle Rechner bereit stellt. Auf diesen virtuellen betriebene Software verhält ich wie Software, die direkt auf einem physischen betrieben wird.

WS (Web Services) Ein Transportdienst, der über das Internet eine einfach Integration von Anwendungen auf verschiedenen, physikalisch verteilten Systemen ermöglicht [Heinrich u. a., 2004, 709]. Mit ihnen dient das Internet nicht mehr nur der Kommunikation mit Menschen (Email, Browser), sondern auch von Anwendungen untereinander [Rechenberg / Pomberger, 2006, 1132].

W3C (World Wide Web Consortium) internationales Konsortium, welches aus über 400 Unternehmen (Stand August 2008), eigenen Mitarbeitern und einer öffentlichen Community besteht und Standardisierungen für Internet-Technologien vorantreibt bzw. entwickelt.

ZF (Zend Framework) mächtiges objektorientiertes → Framework für → PHP, entwickelt von der Zend Company, die auch einer der Hauptentwickler von PHP ist.

Literaturverzeichnis

Die Literaturangaben sind alphabetisch nach den Namen der Autoren sortiert. Bei mehreren Autoren wird nach dem ersten Autor sortiert.

[ALEA Produktbroschüre 2009] ALEA Produktbroschüre: *ALEA Commerce Suite. Produktinformationen.* 2009

[ALEA Vertriebspräsentation 2009] ALEA Vertriebspräsentation: *ALEA Commerce Suite. Die neue Branchenlösung für den Multi-Channel Versandhandel.* 2009

[Bange u. a. 2003] Bange, Carsten / Narr, Jörg / Keller, Patrick / Dahnken, Oliver: *Data Warehousing und Datenintegration. 15 Softwarelösungen im Vergleich.* München : Oxygon Verlag 2003

[Batini u. a. 1986] Batini / Lenzerin / Navathe: A Comparative Analysis of Methologies for Database Schema Integration. In: *ACM* (1986), Nr. 18, S. 323–364

[Bien 2007] Bien, Adam: *Java EE 5 Architekturen.* Frankfurt/Main : Entwickler.press 2007

[Bry u. a. 2005] Bry, Francois / Kraus, Michael / Olteanu, Dan / Schaffert, Sebastian: *Semistrukturierte Daten.* 2005. – `http://www.gi-ev.de/no_cache/service/informatiklexikon/informatiklexikon-detailansicht/meldung/semistrukturierte-daten-77.html`. – Letzter Seitenaufruf am 12.03.2009

[Burbiel 2007] Burbiel, Herbert: *SOA & Webservices in der Praxis.* Poing : Franzis 2007

[Conrad 1997] Conrad, Stephan: *Föderierte Datenbanksysteme. Konzepte der Datenintegration.* Berlin : Springer 1997

[Corsten / Gabriel 2004] Corsten, Daniel / Gabriel, Christoph: *Supply Chain Management erfolgreich umsetzen.* 2. Aufl. Berlin, Heidelberg, New York et.al. : Springer 2004

[Eulgem 1998] Eulgem, Stefan (Hrsg.): *Die Nutzung des unternehmensinternen Wissens.* Frankfurt/Main, Berlin, Bern et.al. : Lang 1998 (Europäische Hochschulschriften)

[Gamma u. a. 2003] Gamma, Erich / Helm, Richard / Johnson, Ralph / Vlissides, John: *Entwurfsmuster. Elemente wiederverwendbarer objektorientierter Software.* München : Addison-Wesley 2003

[Heine 1999] Heine, Peter: *Unternehmensweite Datenintegration. Modular-integrierte Datenlogistik in betrieblichen Informationssystemen.* Leipzig : Teubner 1999

[Heinrich u. a. 2004] Heinrich, Lutz J. / Heinzl, Armin / Rothmayr, Friedrich: *Wirtschaftsinformatik-Lexikon.* 7. Aufl. München : Oldenbourg 2004

[Höding 2000] Höding, Michael: *Methoden und Werkzeuge zur systematischen Integration von Dateien in föderierte Datenbanksysteme.* Dissertation, Universität Magdeburg, Aachen, 2000

[Kecher 2006] Kecher, Christoph: *UML 2.0: das umfassende Handbuch.* Bonn : Gallileo Press 2006

[Keller 2002] Keller, Wolfgang: *Enterprise Application Integration. Erfahrungen aus der Praxis.* 2. Aufl. Heidelberg : dpunkt-Verlag 2002

[Kimsal 2008] Kimsal, Mark: *php/Architect's Guide to E-Commerce Programming with Magento.* Toronto : php|Architect 2008

[Koschel u. a. 2006] Koschel, Arne / Fischer, Stefan / Wagner, Gerhard: *J2EE, Java EE kompakt. Enterprise Java: Konzepte und Umfeld.* München : elsevier 2006

[Leser / Naumann 2007] Leser, Ulf / Naumann, Felix: *Informationsintegration. Architekturen und Methoden zur Integration verteilter und heterogener Datenquellen.* Heidelberg : dpunkt-Verlag 2007

[Lipinski 2001] Lipinski, Klaus (Hrsg.): *Lexikon der Datenkommunikation.* 6. Aufl. Bonn : mitp 2001

[Lohse 2003] Lohse, Anja: *Integration unterschiedlich strukturierter Daten.* Dissertation, TU Dresden, Köln, 2003

[Mertens / Griese 1993] Mertens, Peter / Griese, Joachim: *Integrierte Informationsverarbeitung. Administrations- und Dispositionssysteme in der Industrie.* Wiesbaden : Gabler 1993

[OASIS 2002] OASIS ; Bellwood, Tom (Hrsg.) / Clément, Luc (Hrsg.) / Ehnebuske, David (Hrsg.) / Hately, Andrew (Hrsg.) / Hondo, Maryann (Hrsg.) / Husband, Yin L. (Hrsg.) / Januszewski, Karsten (Hrsg.) / Lee, Sam (Hrsg.) / McKee, Barbara (Hrsg.) / Munter, Joel (Hrsg.) / Riegen, Claus von (Hrsg.): *UDDI Version 3.0 - UDDI Spec Technical Committee Specification, 19 July 2002.* 2002. – `http://uddi.org/pubs/uddi-v3.00-published-20020719.htm`. – Letzter Seitenaufruf am 21.03.2009

[Ralev 2004] Ralev, Susanna: *Betriebswirtschaftliche Konzepte für das Supply Chain Management in der SAP Software.* Diplomarbeit, Hochschule Mittweida (FH), Mittweida, 2004

[Rechenberg / Pomberger 2006] Rechenberg, Peter / Pomberger, Gustav: *Informatikhandbuch.* 4. Aufl. München, Wien : Hanser 2006

[Reichling 2001] Reichling, Helmut: *Elektronische Marktplätze. Chancen für den Mittelstand.* S. 93–108. In: *Die Supply Chain im Zeitalter von E-Business und Global Sourcing.* Paderborn : Fraunhofer-Anwendungszentrum für Logistikorientierte Betriebswirtschaft 2001 (Fraunhofer ALB-HNI-Verlagsschriftenreihe)

[Schneider / Werner 2007] Schneider, Uwe / Werner, Dieter: *Taschenbuch der Informatik.* 6. Aufl. Leipzig, München : Fachbuchverlag Leipzig im Carl-Hanser-Verlag 2007

[Schulz 2009] Schulz, Sebastian: *Surfverhalten und -Erleben in Webhops.* Dissertation, Universität Göttingen, Hamburg, 2009

[Schwarze 2000] Schwarze, Jochen: *Einführung in die Wirtschaftsinformatik.* 5. Aufl. Herne, Berlin : Verlag Neue Wirtschaftsbriefe 2000

[Sklar / Trachtenberg 2005] Sklar, David / Trachtenberg, Adam: *PHP5 Kochbuch.* 2. Aufl. Beijing, Cambridge, Farnham et.al. : O'Reilly 2005

[SoftGuide GmbH & Co. KG 2009] SoftGuide GmbH & Co. KG: *ALEA Commerce Suite. Neue Branchenlösung für Versandhandel.* 2009. – `http://www.softguide.de/prog_a/pa_1304.htm`. – Letzter Seitenaufruf am 16.04.2009

[Speyerer 2005] Speyerer, Jochen: *Flexible Integration der Informationsverarbeitung im Supply Chain Management mit Web Services unter besonderer Berücksichtigung von Logistikdienstleistern.* Dissertation, . . . , Berlin, 2005

[Stockbauer 2008] Stockbauer, Markus: Magento im Praxiseinsatz. In: *T3N* 5 (2008), Nr. 13, S. 32–35

[Strobel 2006] Strobel, Thorsten: *Internetbasierte Datenintegration für Automatisierungssysteme.* Dissertation, TU Stuttgart, Stuttgart, 2006

[Techdivision 2009] Techdivision: *Magento Benutzerhandbuch.* 2. Aufl. Techdivision 2009

[Thome 2006] Thome, Rainer: *Grundzüge der Wirtschaftsinformatik.* München : Pearson Studium 2006

[Voigtmann / Zeller 2002] Voigtmann, Peter / Zeller, Thomas: *Enterprise Application Integration und B2B Integration im Kontext von Electronic Business und Elektronischen Marktplätzen. Teil1: Grundlagen und Anforderungen.* Nürnberg : FORWIN-Bericht, FWN-2002-013 2002

[W3C 2001] W3C: *Web Services Description Language (WSDL) 1.1 – W3C Note 15 March 2001.* World Wide Web Consortium (Veranst.), 2001. – `http://www.w3.org/TR/2001/NOTE-wsdl-20010315`. – Letzter Seitenaufruf am 19.02.2009

[W3C 2003a] W3C: *SOAP Version 1.2 Part 1: Messaging Framework – W3C Recommendation 24 June 2003.* World Wide Web Consortium (W3C) (Veranst.), 2003. – `http://www.w3.org/TR/2003/REC-soap12-part1-20030624`. – Letzter Seitenaufruf am 19.02.2009

[W3C 2003b] W3C: *SOAP Version 1.2 Part 2: Adjuncts – W3C Recommendation 24 June 2003.* World Wide Web Consortium (W3C) (Veranst.), 2003. – `http://www.w3.org/TR/2003/REC-soap12-part2-20030624`. – Letzter Seitenaufruf am 19.02.2009

[W3C 2004a] W3C: *Web Services Architecture. W3C Working Group Note 11 February 2004.* World Wide Web Consortium (W3C) (Veranst.), 2004. – `http://www.w3.org/TR/2004/NOTE-ws-arch-20040211`. – Letzter Seitenaufruf am 19.02.2009

[W3C 2004b] W3C: *XML Schema Part 1: Structures Second Edition - W3C Recommendation 28 October 2004.* World Wide Web Consortium (Veranst.), 2004. – `http://www.w3.org/TR/2004/REC-xmlschema-1-20041028`. – Letzter Seitenaufruf am 21.02.2009

[Wegweiser GmbH 2007] Wegweiser GmbH ; Lorenz, Oliver (Hrsg.): *eBusiness 2007/2008. Jahrbuch der deutschen Wirtschaft.* 2007

[Welling / Thomson 2009] Welling, Luke / Thomson, Laura: *PHP5.3 und MySQL5.1. Dynamische Webanwendungen von Einstieg bis E-Commerce.* München : Markt + Technik 2009

[Wenzel u. a. 2001] Wenzel, Rüdiger / Fischer, Georg / Metze, Gerhard / Nieö, Peter: *Industriebetriebslehre – Das Management des Produktionsbetriebs.* München, Wien : Fachbuchverlag Leipzig im Carl-Hanser-Verlag 2001

[Willkommer 2008] Willkommer, Josef: E-Commerce mit Open Source. In: *T3N* 5 (2008), Nr. 13, S. 26–29

[Wittmann u. a. 2000] Wittmann, Thomas / Hunscher, Matthias / Kischka, Peter / Ruhland, Johannes: *Data Mining. Entwicklung und Einsatz robuster Verfahren für betriebswirtschaftliche Anwendungen.* Frankfurt/Main : Lang 2000

[Zenner 2009] Zenner, Roman: *Online-Shops mit Magento.* Beijing, Cambridge, Farnham et.al. : O'Reilly 2009

Selbständigkeitserklärung

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe. Alle Teile, die wörtlich oder sinngemäß einer Veröffentlichung entstammen, sind als solche kenntlich gemacht.

Die Arbeit wurde noch nicht veröffentlicht oder einer anderen Prüfungsbehörde vorgelegt.

Leipzig, den 20.05.2009 .....